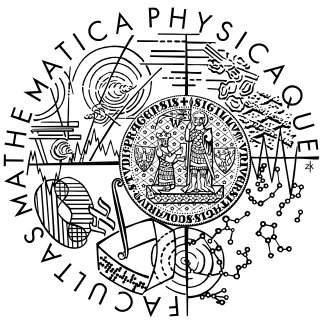
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCA



Martin Choma

Integrace systémů pomocí společného doménového modelu

Katedra softwarového inženýrství

Vedoucí diplomové práce: doc. Ing. Karel Richta, CSc.

Studijní program: Informatika

2009

Ďakujem doc. Ing. Karlu Richtovi, CSc. za vedenie práce, kolegom z Českej poisťovne za ochotu a odborné pripomienky počas našej spolupráce a v neposlednom rade aj mojej rodine a priateľke Kristýne za sústavnú podporu.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 7.8.2009 Martin Choma

Obsah

Názov práce: Integrace systémů pomocí společného doménového modelu
Autor: Martin Choma
Katedra (ústav): Katedra softwarového inženýrství
Vedúci diplomovej práce: doc. Ing. Karel Richta, CSc.
e-mail vedúceho: richta@ksi.mff.cuni.cz

Abstrakt: Kanonický dátový model (Canonical Data Model) obsahuje popis všetkých dát, ktoré sa v organizácií používajú. Všepokrývajúci model však môže byť náročný na realizáciu a preto sa zvyčajne rozsah tohto modelu definuje tak, aby pokrýval dátové modely integrovaných aplikácií. Použitie architektúry založenej na CDM nie je jednoduché. Prvým problémom, ktorému sa práca venuje je definovanie vlastného spoločného dátového modelu. CDM je typicky vytvorený v niektorom z modelovacích nástrojov a popísaný modelovacím jazykom ako napríklad UML. Práca sa ďalej zaoberá vytvorením mapovania medzi formátmi aplikácií a CDM. Táto časť približuje aké transformácie pripadajú do úvahy, ako sú technicky realizovateľné a kde sa môžu nachádzať. Nakoniec je v práci popísaný proces riadenia zmien a dodávok, čo je kľúčová otázka, keďže zmenou CDM môžu byť ovplyvnené všetky aplikácie využívajúce CDM. Je preto potrebné byť na túto situáciu pripravený a rozmyslieť si postup nasadenia najmä nekompatibilných zmien.

Kľúčové slová: kanonický dátový model, CDM, integrácia, UML, SOA

Title: System integration using common data model
Author: Martin Choma
Department: Department of Software Engineering
Supervisor: doc. Ing. Karel Richta, CSc.
Supervisor's e-mail address: richta@ksi.mff.cuni.cz

Abstract: Canonical Data Model contains description of all data used by organization. Such "all-covering" model is difficult to create and therefore it is preferred to define CDM as data model used by messages and events of integration purpose. Using CDM is not simple. First of all it is necessary to define own canonical data model. Typically, CDM is created in some CASE tool and described by some modeling language, e.g. UML. Next issue is mappings and transformation between CDM and application formats. Work lists some kinds of transformations, which can be used, possible implementations and locations of them. Finally work describes change and release management process, which is key part of CDM successful story. CDM change can affect all integrated applications, therefore is important to be ready for this situation and think about deployment process, especially when there are incompatible changes.

Keywords: canonical data model, CDM, integration, UML, SOA

Kapitola 1

Úvod

Každá veľká spoločnosť s dlhou históriou sa musí vysporiadať s veľkým množstvom existujúcich aktívnych aplikácií. Nástup moderných technológií, najmä v oblasti integrácií informačných systémov, umožňuje riešiť obchodné požiadavky na IT systémy pomocou spolupráce dvoch alebo viacerých existujúcich informačných systémov. Z tohto dôvodu sú v rôznych informačných systémoch definované integračné rozhrania, ktoré túto spoluprácu umožňujú. Preto, aby integrácia informačných systémov nebola riešená formou tzv. point-to-point integrácie, teda integrácie, keď systémy vystavujú svoje rozhrania špecificky pre každého nového zákazníka (iný systém) a keď zmena v rozhraní spôsobuje nutnosť zmeny v systémoch, ktoré dané rozhrania využívajú, sa nasadzuje tzv. integračný broker, niekedy tiež označovaný ako middleware.

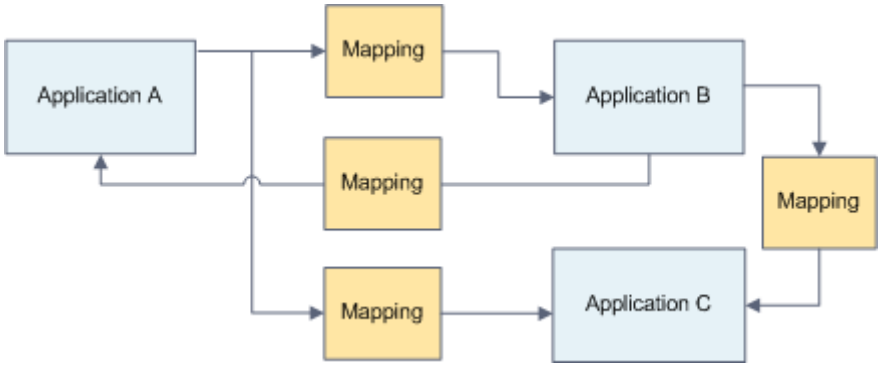
Samotný integračný broker však nie je riešením náročnej situácie v oblasti informačných systémov veľkých spoločností. Integračný broker je nutné doplniť o špecifické metodiky a pravidlá pre budovanie nových rozhraní a procesy spojené so zmenovým manažmentom a manažmentom verzií (change a release management). Jednou z najdôležitejších komponent integračného brokera je tzv. kanonický dátový model (Canonical Data Model, CDM), ktorý zabezpečuje unikátnosť vystavovaných rozhraní a definuje jednoznačný význam dát, ktoré sa pri integrácií prenášajú.

1.1 Motivácia použitia CDM

Ak si vezmeme systémy komunikujúce v uzatvorenom prostredí, napríklad v rámci jednej organizácií, zistíme, že si vymieňajú veľmi podobné správy. Neustále si predávajú údaje o klientovi, jeho kontaktné alebo platobné údaje, informácie o zakúpených produktoch, službách a podobne. Tieto správy majú pre zúčastnené systémy spoločný význam, na ktorom sa zhodnú - ak posielame údaje o klientovi z jedného systému do druhého, obe ju interpretujú podobným spôsobom. Avšak typicky každá aplikácia vyjadruje tieto významovo podobné dáta v inej podobe. Môže byť použitá rozdielna štruktúra entity, rozdielny formát dátumu, rozdielne dátové typy alebo rozdielne identifikátory. Preto je nutné vytvoriť mapovanie medzi oboma doménovými modelmi.

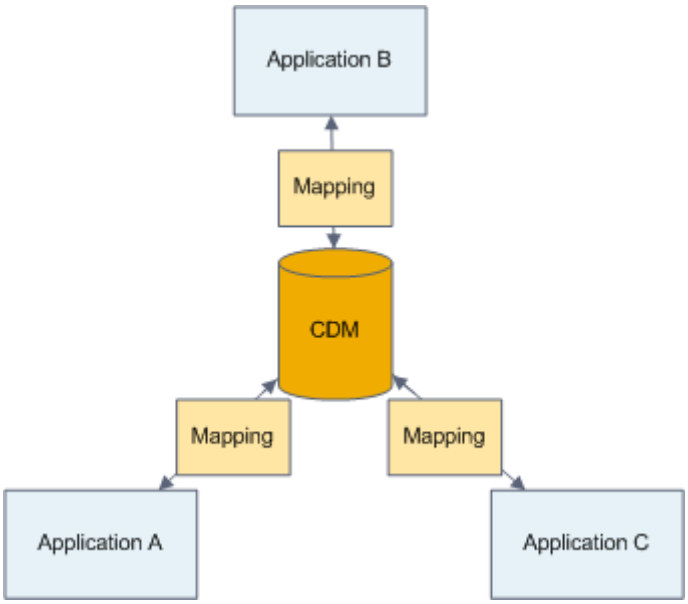
Obrázok 1.1

Časom sa však počet mapovaní medzi jednotlivými aplikáciami môže nepríjemne zvýšiť, čím sa môže vytvoriť ťažko udržiavateľná sieť závislostí.



Obrázok 1.2

Jedným z riešení ako tejto situácií predchádzať je vytvoriť spoločný doménový model, ako jednotný jazyk aplikácií, skrz ktorý by komunikovali. Ak by si každý systém vytvoril svoje mapovanie na CDM, mohli by navzájom komunikovať všetky systémy.



Obrázok 1.3

Cieľom mojej práce bude popísať postup vytvorenia a použitia integračného rozhrania za pomocí spoločného doménového modelu a poskytnúť prototypové riešenie.

1.1.1 Príklad zo života

Ako pekný ilustračný príklad zo života, na ktorý by sa dali s úspechom aplikovať idey CDM, môžeme použiť plenárne zasadnutie Európskeho parlamentu.

Pri plenárnych zasadnutiach, na ktorých sa simultánne tlmočí zo všetkých a do všetkých úradných jazykov Únie, pracuje v priemere 800 až 1 000 tlmočníkov. V zásade platí, že každý tlmočník tlmočí z východiskového do svojho materinského jazyka. Pri 506 možných jazykových kombináciách (23 krát 22 jazykov) nie je však vždy jednoduché nájsť osobu schopnú tlmočiť z určitého jazyka do niektorého iného jazyka. V takom prípade sa používa systém tzv. pilotných jazykov, ktorý spočíva v tlmočení z jedného jazyka do iného, pričom sa prechádza cez tretí jazyk. [5]

Ak si v tomto modeli predstavíme analógiu integračného problému, a teda delegátov z jednotlivých krajín budeme na chvíľku považovať za systémy, ich jazyk za formát dát, ktorý používajú a o tlmočníkov budeme hovoriť ako

o transformačnej službe, mame tu pomerne neefektívny model point-to-point integrácie medzi 23 systémami, ktorý si pre svoj chod vyžaduje vytvorenie a údržbu 506 transformácií.

Ak by sme chceli v tomto prípade aplikovať myšlienky CDM, tak zvolíme jeden jazyk (napr. esperanto alebo angličtinu), do ktorého sa budú prekladať vystúpenia všetkých delegátov. Pri takomto modeli si vystačíme s 23 tlmočníkmi, keďže každý bude prekladať do a zo svojho materinského jazyka.

Výhodou takéhoto prístupu je samozrejme zníženie nákladov. Pri vstupe ďalšieho jazyka, nepotrebujeme tlmočníkov pre každý z existujúcich jazykov, ale bude postačovať jeden.

1.2 Štruktúra práce vo vzťahu k vytýčeným cieľom

Cieľom mojej práce je analyzovať a navrhnúť integračné riešenie pomocou kanonického dátového modelu. Budem sa teda snažiť podať zhrňujúcu správu o možnostiach integrácie pomocou CDM za súčasného stavu, uvediem na čo je potrebné pri zavádzaní CDM v podnikovom prostredí myslieť a akú podporu nám pre tieto účely ponúkajú voľne dostupné nástroje.

V prvej, teoretickej časti sa pokúsim v krátkosti priblížiť oblasť integrácie podnikových systémov s ohľadom na použitie CDM.

V ďalšej analytickej časti rozoberiem jednotlivé kroky životného cyklu integračného riešenia. Od vzniku CDM, jeho udržiavanie až po nasadenie výslednej služby a realizáciu zmenových požiadaviek.

A v záverečnej praktickej časti už na praktickej ukážke predvediem jeden z možných spôsobov ako by mohlo výsledné riešenie vyzerať.

Kapitola 2

Terminológia

2.1 Integrácia

Pod pojmom integrácie rozumieme prepojenie dvoch a viacerých systémov, za účelom zdieľania informácii alebo poskytovania funkčností.

Integráciu komplikuje to, že často ide o spájanie heterogénnych prostredí; systémy bežia fyzicky na rôznych počítačoch, s odlišnou konfiguráciou, sú tvorené pomocou rozličných technológií a využívajú rozdielne doménové modely.

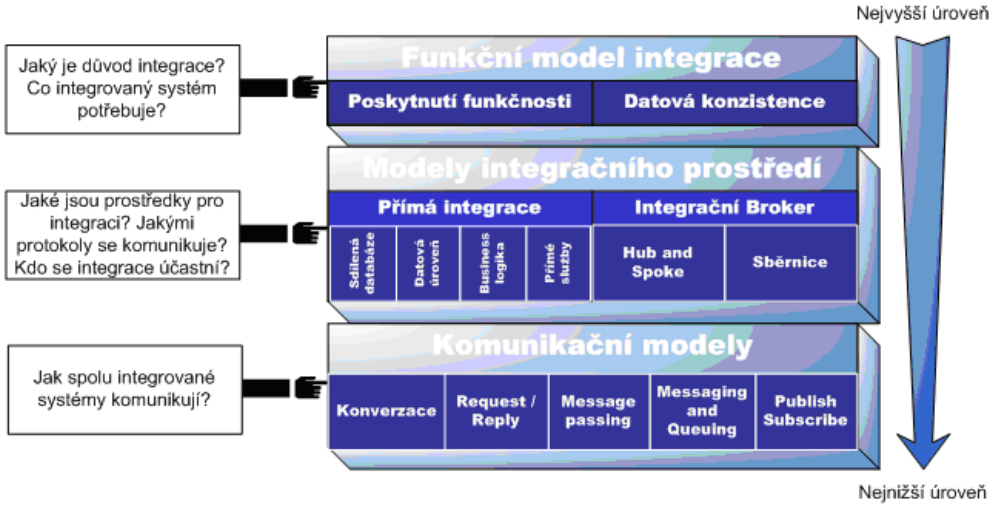
Hovoríme, že systém poskytuje *službu*, ak poskytuje funkčnosť alebo dáta. *Producent* je systém, ktorý pre záujemcov poskytuje službu. *Konzumentom* rozumieme systém využívajúci službu producenta. Ak hovoríme o *tesnej väzbe* (tightly-coupled) dvoch systémov máme na mysli takú väzbu, pri ktorej platí, že systémy sú na sebe závisle a to napríklad buď zdieľaním kódu alebo informáciou o umiestnení. Naopak *voľná väzba* (loosely-coupled) systémov znamená, že systémy nie sú na sebe závislé.

Integračná cesta je cesta, po ktorej prebieha komunikácia medzi producentom a konzumentom. *Integrační broker* (middleware) je sprostredkovateľ integračnej cesty (monitoring, smerovanie, transformácie…). *Synchrónna integrácia* je integrácia kedy systém, ktorý je iniciátorom integrácie vyžaduje odpoveď od integrovaného systému v rámci jedného volania, pričom iniciátor očakáva, že jeho požiadavka bude riešená okamžite a do doby prijatia odpovede je blokovaný. Pri *Asynchrónnej integrácii* systém, ktorý je iniciátorom integrácie nevyžaduje odpoveď od integrovaného systému v rámci jedného volania, pričom iniciátor očakáva, že jeho požiadavka nebude riešená

okamžite, ale v dobe, ktorú si určí sám integrovaný systém. Iniciátor nie je blokovaný.

Integráciu informačných systémov je možné rozdeliť podľa rôznych pohľadov, tri základné sú:

- Funkčný pohľad
- Pohľad integračného prostredia
- Komunikačný pohľad



Obrázok 2.1: Rozdelenie integrácie podľa modelov [20]

2.1.1 Funkčný pohľad

Z funkčného hľadiska je možné integračné riešenia rozdeliť do dvoch skupín

- Poskytnutie služby - myslí sa tým situácia, keď konzument vyžaduje ad-hoc informáciu alebo funkčnosť od producenta.
- Konzistencia dát – alebo inými slovami replikácia dát je prípad, kedy je požadované, aby dva alebo viac systémov zdieľali časť svojej bázy dát za účelom ďalšieho spracovania.

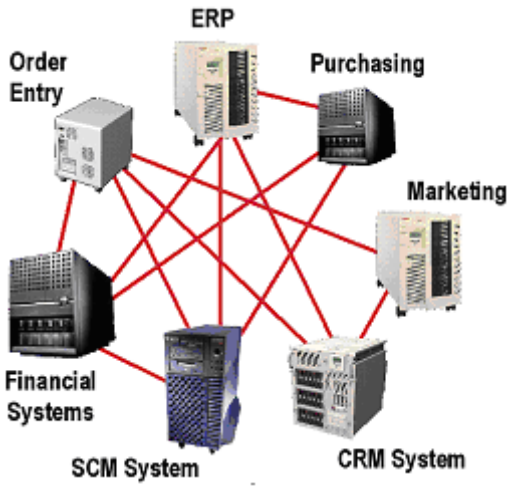
2.1.2 Pohľad integračného prostredia

Čo sa týka infraštruktúry, rozlišujeme dva typy integrácie

- Priama integrácia - Jedná sa o integráciu medzi práve dvomi konkrétnymi systémami bez integračného brokera.
- Integrácia skrz integračný broker – Integračný broker je softwarový komponent, ktorý zaradením medzi účastníkov integrácie (v roli hubu alebo zbernice) a vhodným použitím môže zabrániť vzniku tesnej väzby medzi systémami.

Priama integrácia

Taktiež známa pod pojmom integrácia point-to-point sa využívala prevažne pred nástupom systémov typu integračný broker. Dôsledkom takéhoto prístupu je integračný zmätok ťažko dokumentovateľných a udržiavateľných väzieb, ktorý sťažuje rozvoj a nasadenie nových verzií aplikácie. Tento typ integrácie sa nedá efektívne riadiť a je ho ťažké monitorovať, či auditovať. Ďalšou nevýhodou je, že integrácia je netransparentná a dopad na ďalšie systémy je ťažko predvídateľný, čo pri prípadnej zmene prináša riziko neskorého odhalenia nefunkčnosti a nutnosti návratu na predchádzajúcu verziu.



Obrázok 2.2: Fiktívny príklad priamej integrácie [20]

Zdieľaná databáza

tiež je možné naraziť na pojem integračná databáza (integration database [6]). Integračná databáza je databáza, ktorá sa chová ako úložište dát pre viacero aplikácií a tým pádom integruje obidve aplikácie. Schéma takejto databázy musí brať do úvahy požiadavky všetkých klientov. Výsledná schéma je

zvyčajne obecnejšia, zložitejšia alebo oboje naraz. Zmeny do schémy musia byť odsúhlasené všetkými klientmi.

Integrácia na dátovej úrovni

Deje sa tak napríklad pomocou DB linku. Systémy sa takouto synchronizáciou stávajú závislé, vzniká medzi nimi tesná väzba. Konzument musí poznať presnú konfiguráciu spojenia na producenta. (adresa, login, heslo). A hlavne integrácia je založená na vzájomnej znalosti databázových schém, čo prináša zjavné problémy pri údržbe.

Integrácia na úrovni aplikačnej logiky (vzdialené volanie procedúr)

Jedná sa o synchrónnu integráciu typu žiadosť/odpoveď (request/reply). Pri ktorej konzument kontaktuje producenta pomocou platformovo-špecifického protokolu. Napríklad v prípade Javy je to RMI, alebo .NET Remoting v prípade konkurenčnej platformy. Takýto typ integrácie je vhodný v prípadoch, kedy sú veľké nároky na výkonnosť, prípadne sa vyžaduje aby bolo volanie súčasťou distribuovanej transakcie. Integrované aplikácie zdieľajú časť svojich objektov (v knižniciach) a konzument musí udržiavať prístupové údaje (adresa, login, heslo) čím medzi nimi vzniká tesná väzba.

Integrácia na úrovni priamo volaných služieb

Jedná sa o rozšírenú variantu integrácie na úrovni aplikačnej logiky, v ktorej je aplikačná logika vystavená pomocou rozhrania webovej služby a prenos prebieha pomocou protokolu SOAP. Na rozdiel od predchádzajúcich troch riešení systémy komunikujú pomocou štandardného rozhrania webových služieb a protokolom SOAP, nevymieňajú si medzi sebou celé objekty, ale XML dokumenty. Webovú službu je teda možné znovu použiť aj pre inú platformu. Na druhú stranu je takéto volanie pomalšie ako v prípade RMI volania a nedá sa zaradiť do distribuovanej transakcie. A naďalej platí nevýhoda plynúca z tesnej väzby a akékoľvek zmeny sa musia synchronizovať na oboch stranách. To sa môže na strane producenta vyriešiť nasadením novej verzie služby a súčasným udržiavaním pôvodnej verzie, čo však nie je žiaduce.

Integrácia skrz integračný broker

Použitie integračného brokera vyplýva zo snahy minimalizovať zásahy do viacerých aplikácií pri zmene jedného rozhrania. Integračný broker činí aplikácie väčšmi nezávislé, poskytuje lepšie možnosti škálovania a dodržovania štandardov.

Usporiadanie Hub-and-Spoke

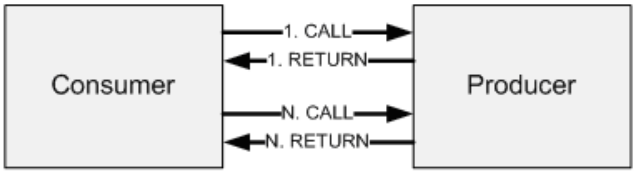
V tomto usporiadaní je medzi jednotlivé systémy vložená dodatočná vrstva v podobe integračného hubu. Každé dva integrované systémy, tak nekomunikujú medzi sebou priamo, ale práve prostredníctvom hubu. Integrácia s novými systémami sa stáva jednoduchšia a zaručuje centralizáciu mnohých funkcií; centrálne monitorovanie, audit, bezpečnosť, transformácie, spoľahlivé doručenie správ, prekonanie výpadku systému atď. Zároveň je väzba medzi systémami voľnejšia, čo prináša väčšiu flexibilitu celého prostredia a možnosť náhrady systémov v budúcnosti. Príkladom takéhoto usporiadania je zbernica podnikových služieb (Enterprise Service Bus).

2.1.3 Komunikačný pohľad

Komunikačný pohľad popisuje spôsob akým spolu systémy komunikujú, tj. akým smerom tečú informácie, kto je iniciátorom integrácie atď.. Principiálne rozlišujeme päť základných komunikačných pohľadov.

Konverzační model

Aplikácie spolu komunikujú formou dialógu, pričom každý krok dialógu môže byť ovplyvnený predchádzajúcim krokom, tzn. že sa jedná o stavovú komunikáciu. Každá konverzácia je jednoznačne označená unikátnym identifikátorom, pre ktorý je minimálne na strane producenta udržovaný konverzačný kontext. Jedná sa o synchrónny, on-line, obojsmerný komunikačný model. Konverzačný model nie je doporučený, pretože rozhranie postavené na konverzačnom modeli sa dá len s ťažkosťami generalizovať a nutnosť údržby konverzačného kontextu na strane konzumenta alebo producenta výrazne komplikuje implementáciu a zvyšuje tak náklady na integráciu.



Obrázok 2.3

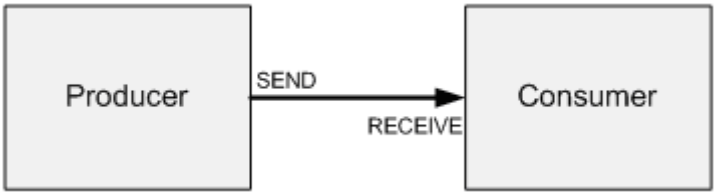
Request/Reply model

Request/Reply model je určený k jednorazovému volaniu funkčnosti (služby) inej aplikácie. Jednému volaniu vždy zodpovedá jedna odpoveď. Medzi jednotlivými volaniami nie je udržovaná väzba, tzn. že sa jedna o bezstavovú komunikáciu. Najčastejšie sa realizuje pomocou webových služieb. Jedná sa o synchrónny, on-line, obojsmerný komunikačný model.

Obrázok 2.4

„Message passing“ model

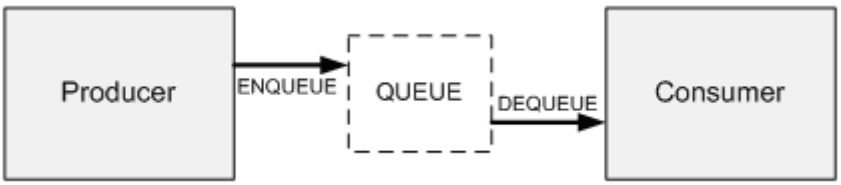
„Message passing“ model je určený k zverejňovaniu udalostí zmien medzi dvoma systémami. Producent na základe identifikovanej udalosti generuje správu, ktorú zasiela jednému alebo viacerým konzumentom (v prípade, že je viacero odberateľov, je nutné, aby boli u producenta registrovaní), pričom nečaká na spracovanie udalosti na strane konzumenta. Jedná sa o on-line, asynchrónny, jednosmerný komunikačný model. Tento model nie je doporučený, pretože aj keď je asynchrónny, vyžaduje, aby bol konzument v dobe integrácie dostupný, čím výhodu asynchrónnosti stráca. Namiesto „message passing“ modelu je doporučené využiť model „message queuing“ (viď ďalej).



Obrázok 2.5

„Message queuing“ model

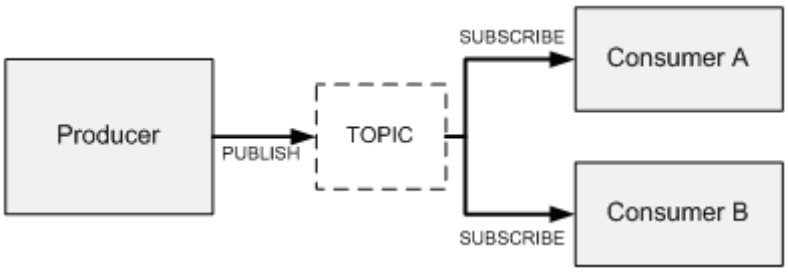
Je určený primárne k zverejňovaniu udalostí zmien medzi dvoma systémami. Producent nekomunikuje priamo s integrovanou aplikáciou, ale namiesto toho kontaktuje integračnú vrstvu, ktorá správu od producenta uloží do internej fronty, kde je pripravená k vyzdvihnutiu konzumentom. Nie je nutné, aby bol konzument v dobe odosielania správy producentom dostupný. Konzument si z fronty odoberá správy podľa toho, ako je aktuálne vyťažený. Nutnosť údržby a správy fronty na integračnej vrstve. Jedná sa o semi-online, asynchrónny, jednosmerný komunikačný model.



Obrázok 2.6

„Publish and subscribe“ model

je variáciou „message queuing“ modelu. Na strane integračnej vrstvy je definovaný tzv. topik (topic), do ktorého producent v prípade vzniku udalosti zasiela správu. K topiku sú zaregistrovaní odberatelia (konzumenti), ktorí správy odoberajú. Každá správa je prístupná pre všetkých zaregistrovaných odberateľov. Jedná sa o semi-online, asynchrónny, jednosmerný komunikačný model. Tento model je doporučený pre dátové synchronizácie, kde je známe, alebo sa dá odhadnúť, že o dáta producenta má záujem viacero konzumentov.



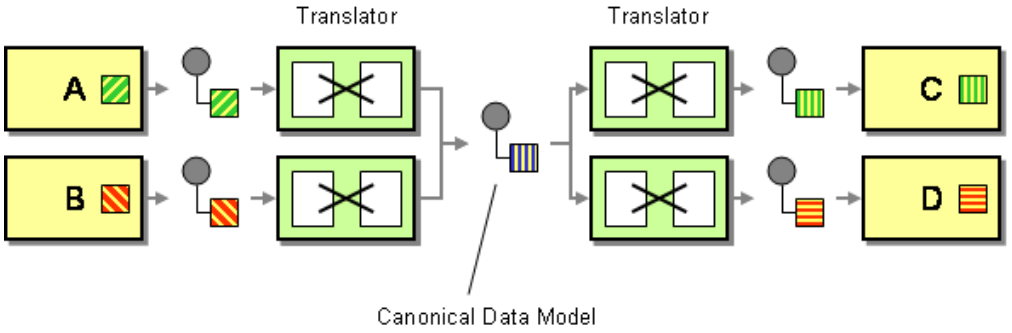
Obrázok 2.7

2.2 CDM

To čo sa v tejto práci skrýva pod skratkou CDM predstavil Gregore Hophe vo svojej knihe Enteprise Integration Patterns [9] ako integračný návrhový vzor kanonický dátový model (Canonical Data Model). Popisuje ho následovne;

Problém; Integrujeme niekoľko aplikácií pomocou vzájomného zasielania správ (messaging), každá využíva svoj doménový model. Ako minimalizovať previazanosť aplikácií, ktoré využívajú rozdielne dátové formáty?

Riešenie; Zavedieme kanonický dátový model (Canonical Data Model) ako ďalšiu vrstvu indirekcie medzi aplikáciami. Ak je pridaná nová aplikácia do integračného riešenia, stačí pridať transformáciu medzi kanonickým modelom a samotnou aplikáciou.



Obrázok 2.8: Schéma integračného vzoru kanonický dátový model [9]

Kanonický indikuje, že pôjde o jednoznačný model - istým spôsobom normalizovaný pre potreby organizácie a tým pádom referenčný a všeobecne uznávaný.

Tak ako v prípade najznámejších návrhových vzorov skupiny „GoF“ (Gang of Four) [8] aj integračné návrhové vzory [9] by mali slúžiť ako spoločný dorozumievací prostriedok, v tomto prípade, systémových integrátorov. Nie každý prijal tento termín a preto je v tejto súvislosti možné naraziť aj na iné pojmy označujúce túto integračnú konštrukciu ako spoločný dátový model (Common Data Model), či kanonický doménový model (Canonical Domain Model) alebo aj centrálny doménový model (Central Domain Model).

Ja osobne by som prívlastok „dátový“ zamenil za „doménový“, keďže pojem „dátový“ je spätý s databázovým svetom, pričom pojem „doménový“ je bližší

objektovým jazykom, ktoré sa v integrácií pomocou zasielania správ využívajú najviac. Zároveň „doménový“ sa viac približuje predstave CDM ako vyjadrovaciemu prostriedku istej obchodnej problematiky – obchodnej domény.

V tejto práci sa ďalej bude používať najrozšírenejší a najpoužívanejší termín kanonický dátový model.

2.2.1 Prínosy CDM

Zavedenie CDM prináša tieto výhody:

- Použitie kanonického modelu odtieňuje jednotlivé systémy od formátu dát iných systémov, tým sa zvyšuje ich miera nezávislosti od okolitých aplikácií.
- Vytvorenie kanonického modelu je jedinečnou príležitosťou na zadefinovanie katalógu jednoznačných pojmov v rámci spoločnosti. Čo prináša výhody v podobe existencie referenčného doménového modelu, s ktorým môžu jednotlivé aplikácie konzultovať svoj doménový model. Prípadne zavedenie jednotného slovníku pojmov, čo uľahčí komunikáciu pracovníkov [10].
- Z technického hľadiska prináša tento integračný vzor výhodu, keďže na koncových bodoch je potrebná iba jedna transformácia. Konzument sa pripája iba k jednému typu dát, bez ohľadu, či je zdrojov viacero alebo nie [11].
- Použitie medzivrstvy kanonického formátu zabezpečuje nezávislosť transformačných služieb od producenta. Bez takejto vrstvy by si zmena na strane producenta vynútila zmenu transformácií konzumentov.
- Ak by sa transformácia vykonávala priamo medzi jednotlivými lokálnymi formátmi, mohlo by teoreticky dôjsť ku kombinácií prepojenia všetkých aplikácií. To vedie k zvyšovaniu správy a úsilia na udržovanie

- Existujúci kanonický model určuje jednoznačný slovník entít vo vnútri firmy s podobnou štruktúrou bez ohľadu na to, kde vznikajú. Zjednodušuje komunikáciu vývojárov a urýchľuje vývoj nových systémov, tým že poskytuje overený referenčný model, z ktorého je možné vychádzať a vyvarovať sa tak nesprávnym rozhodnutiam, ktoré môžu viesť k ťažšej údržbe systémov.

2.2.2 Čo CDM vytýkajú kritici

Kritici CDM vytýkajú tieto nedostatky:

- *„Môže sa stať veľkým molochom, ktorému nebude nikto rozumieť“* [21]

  Áno, tak ako akýkoľvek softvérový artefakt. Pri tvorení CDM sa musí ctiť princíp zrozumiteľnosti a kvalitnej dokumentácie. Treba však povedať, že nie je rozumné pokúšať sa v CDM zahrnúť všetky biznis entity spoločnosti. CDM je zamýšľané pre použitie v integračnej doméne, preto je postačujúce v CDM udržiavať len entity, ktoré sa zúčastňujú integrácie.

- *„Je to neefektívne, pri integračnom riešení sa vytvorí o jedno mapovanie viac ako bez CDM“*

  To je ovšem krátkozraký pohľad. Naopak očakáva sa znovupoužitie mapovaní a výsledný menší počet transformácií.

- *„O(n^2) mapovania sa v praxi nevyskytuje“* [21]

  Áno, nevyskytuje sa, ale nech je to iba O(2n), alebo O(3n), už vtedy začína byť otázka znovupoužitia transformácií zaujímavá.

- *„Neexistuje jednotný pohľad a chápanie biznis entít.“*

  Ak máme integrovať dva systémy, tak si nutne musia pohľad na biznis entity zjednotiť, ináč by sa nedohovorili. Tento spoločný pohľad potom môžeme použiť v CDM.

2.2.3 Čo CDM nie je

Keďže nie je jednoznačne definovaná terminológia, stáva sa, že dochádza k skresleným predstavám o tom, čo CDM je, resp. nie je. Na tomto mieste explicitne vymenujem, s čím by sa dalo CDM zameniť, ale nie je obsahom tejto práce.

Aplikačný doménový model

V prípade, že je reč o jednotnom doménovom modeli (Common Domain Model) alebo centrálnom doménovom modeli (Central Domain Model), hrozí nebezpečenstvo, že sa vybaví predstava o doménovom modeli na úrovni business logiky aplikácií, ktorý by si mala každá aplikácia osvojiť.

Realizovať takú ideu by bolo príliš pracné, respektíve drahé, v prostredí, kde existuje množstvo heterogénnych aplikácií, starých alebo nových, ktoré boli vytvorené s doménovými modelmi podľa uváženia každého implementujúceho tímu a kde je taktiež možné nájsť aplikácie tretích strán zakúpených na kľúč, samozrejme s vlastným doménovým modelom, nereflektujúcim zvyklosti danej firmy. Rovnaký prípad predstavujú systémy zdedené zo zlúčených firiem, ak také existujú.

Aj keby sme si predstavili, že by toto bolo realizované, dá sa predpokladať, že údržba takého riešenia by bola neľahkou úlohou a časom by pravdepodobne dochádzalo k rezignácií tímov na synchronizáciu s jednotným doménovým modelom. Ovšem treba pre korektnosť priznať, že podobné argumenty ohľadne obtiažnosti údržby sa dajú použiť proti CDM, tu je však situácia trochu odlišná, keďže CDM nekladie na aplikácie tak prísne obmedzujúce podmienky ako dohoda o používaní spoločného doménového modelu. Každý systém môže použiť svoj vlastný model. Transformačná služba v CDM sa postará o prevedenie dát do a zo spoločného doménového modelu. Komunikujúce systémy komunikujú s integračným brokerom vo vlastnom doménovom jazyku. Samozrejme čím viac je aplikačný doménový model podobný CDM, tým je transformačná služba jednoduchšia.

Spoločné dátové úložište

Predstava jedného obrovského úložišťa - miesta kde by všetky aplikácie ukladali svoje dáta, sa môže vybaviť, keď sa hovorí o spoločnom dátovom modeli „Common Data Model“. CDM nie je v žiadnom prípade úložište, poskytuje iba popis spoločných dát.

2.2.4 CDM a SOA

Servisne orientovaná architektúra je často diskutovanou témou poslednej doby. SOA nie je jednoznačne definovaná a každý si pod pojmom SOA predstaví niečo iné. V zásade ide o architektonický koncept, kde sa vydefinuje sada nezávislých služieb s konkrétnou funkčnosťou, ktoré sú volané jednotlivými aplikáciami podľa potreby. Takáto architektúra kladie dôraz na flexibilitu, znovupoužiteľnosť a voľnú väzbu služieb.

Mnoho organizácií, ktorých IT tvorí stovky aplikácií a služieb však zisťuje, že ich implementácia SOA je príliš zložitá a ťažko sa spravuje. Častou príčinou tohto nepríjemného zistenia je dátová integrácia. Problémom sú rozdiely na úrovni fyzických dát medzi aplikáciami a službami, ktoré sa účastina podnikového procesu [4]. Tieto rozdiely sa typicky riešia ručným mapovaním a to má za následok vytvorenie point-to-point väzby na dátovej vrstve, čo prináša obdobné problémy pri realizácií zmien ako v prípade integračných point-to-point riešení, ktoré SOA nahradila.

Pomocou spoločného dátového modelu je možné definovať spoločný referenční bod pre každú dátovú transformáciu a sledovať jej vplyv na systém ako celok. Architektúra využívajúca CDM podporuje komplexnú agregáciu dát, ktorá môže byť v decentralizovanej architektúre veľmi zložitá. [4]

Ak pri zavádzaní SOA nepoužijeme CDM môže sa nám zvýšiť celková zložitosť riešenia, pretože budú vznikať point-to-point väzby na dátovej úrovni. Správne použité CDM je jednou z častí úspešného adaptovania SOA princípov.

Kapitola 3

Analýza použitia CDM

V tejto analytickej časti, budem hľadať odpovede na otázky, ktoré si musí každá organizácia položiť pri zavádzaní CDM. Od kritérií pre výber CDM cez správu transformácií až po nasadenie zmenových požiadaviek.

3.1 Vytvorenie CDM

Vytvorenie CDM v organizácií je neľahká úloha. CDM model je artefaktom, ktorý ovplyvňuje celú firmu. Jeho vzniku a údržbe sa preto musí klásť veľký dôraz a mali by sa na ňom podieľať technici, analytici aj osoby blízke obchodu, pod vedením kompetentnej osoby. Súčasťou výslednej dodávky by mal byť model s kvalitnou dokumentáciou.

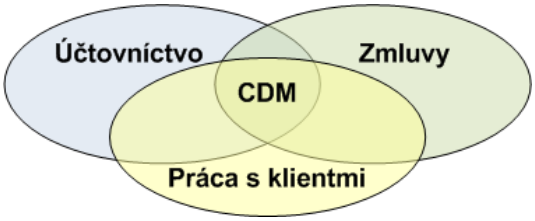
V prípade, že v prostredí kde sa snažíme o vytvorenie CDM existuje dominantný systém, použijeme funkčnosť spätného inžinierstva (reverse engineering), ktorú dnes nájdeme vo väčšine modelovacích nástrojov. Ak takýto systém neexistuje, budeme nútený model vytvoriť od začiatku vlastnými silami.

V nasledujúcej časti sa budem zamýšľať nad tým, čo je možné v CDM udržovať a za akú cenu sa to dá dosiahnuť.

3.1.1 Základné Princípy pri tvorbe CDM

CDM je potrebné budovať postupne podľa aktuálnych potrieb. Je absolútne nevhodné pokúšať sa vytvoriť všetko pokrývajúci model obsahujúci všetky entity obchodnej domény. Je vhodné začať tvoriť CDM určený pre integračné

účely, čiže taký, ktorý bude obsahovať len entity, ktoré sa pri integrácií používajú. Entity, ktoré sa nevymieňajú medzi systémami by nemali byť súčasťou CDM.



Obrázok 3.1

Pred samotným zostavením modelu je potrebné skontrolovať, či už neexistujú modely, ktoré by sa dali použiť ako základ. Napríklad v telekomunikačnom prostredí je populárny model SID, model Acord by sa naopak dal použiť v poisťovníctve.

Na tvorbe modelu je potrebné spolupracovať s osobami, ktoré poznajú danú doménu a uistia nás, že model odpovedá biznis chápaniu.

Je potrebné definovať pravidlá, ktoré bude potrebné dodržovať počas tvorby modelu. Napríklad jasné menné konvencie a vzťahy medzi entitami atď. Ich použitie povedie ku konzistentnému a ľahšie pochopiteľnému modelu. Zároveň je vhodné, ak je model distribuovaný v čitateľnej forme.

Pri navrhovaní CDM modelu je vhodné klásť dôraz na sémantiku rozhrania, teda špecifikovať jednotlivé polia v maximálnej možnej miere. Či už pomocou XSD typov alebo sémantických anotácií.

Nevýhodou CDM je, že neodvratne sa dostaneme do štádia, keď CDM rozhranie bude omnoho väčšie ako by bolo jednoúčelové rozhranie. A teda vývojár bude potrebovať kvalitnú dokumentáciu k tomu, aby pochopil, čo sa konkrétne v tejto službe používa a ktoré dáta sa prenášajú.

3.1.2 CDM a biznis pravidlá

V tejto časti sa zamyslím nad potrebou a možnosťami uchovávať biznis pravidlá v modeli CDM. Budem sa snažiť odpovedať na otázky; Aké sú pozitíva takéhoto prístupu? Na aké problémy narazíme ? Ako sa dajú riešiť?

Definovanie biznis pravidiel v CDM je žiaduce hneď z niekoľkých dôvodov:

- V prvom rade sa zvýši výpovedná hodnota modelu. Model s pravidlami je konkrétnejší, čitateľnejší, ľahšie pochopiteľný.
- Ak vystavíme služby v CDM, ktoré nereflektujú biznis pravidlá, v horšom prípade sa nám „podarí“ pretlačiť nekorektné dáta až do cieľového systému, v lepšom prípade sa dáta zastavia pri validácií cieľového systému a systém odpovie chybou. Takéto volanie, je však zbytočné, keďže táto validácia mohla prebehnúť bez nutnosti volať cieľový systém. Inými slovami s uloženými biznis pravidlami je možné niektoré validácie automaticky uskutočniť už na strane ESB platformy bez nutnosti spoliehať sa na validácie cieľového systému.

Pri písaní biznis pravidiel pre CDM musíme rátať s tým, že biznis pravidlá na CDM nemusia platiť pre všetkých rovnako. Očakáva sa, že CDM entity budú použité v mnohých integračných službách a každý môže mať osobité potreby. Napríklad pole `Partner.bankAccount` môže byť povinné v účtovníckom systéme, ale nemusí byť povinné v CRM systéme.

Jednou z možností ako sa k takejto situácií postaviť je udržiavať v CDM iba tie biznis pravidlá, ktoré platia jednoznačne pre všetkých. Druhou možnosťou je snažiť sa uchovať všetky systémovo-špecifické pravidlá. Osobne by som sa skôr prikláňal k prvej možnosti, keďže modelovacie jazyky, ako napríklad UML, nemá priamu podporu pre túto požiadavku a takáto snaha by zaviedla závislosť CDM na znalosti okolitých systémov, čo je principiálne nežiaduce.

V praxi je možné do modelu pridať biznis pravidlá pomocou typov XML Schema jazyku (XML Schema Definition types) alebo pomocou sémantických anotácií (SAWSDL) [16].

XSD typy

XSD umožňuje obmedziť hodnoty, ktoré daný element môže nadobúdať. Napríklad môžeme definovať, že číselná hodnota elementu `age` musí byť v rozmedzí 0 až 120.

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="140"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Ďalej je možné určiť maximálnu a minimálnu dĺžku reťazca, výčtovým typom špecifikovať hodnoty, ktoré môže element obsahovať. Alebo vydefinovať regulárnym výrazom reťazce, ktoré sú akceptovateľné.

```
<xs:element name="car">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Golf"/>
      <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

3.1.3 CDM a číselníkové hodnoty

K číselníkom sa môžeme zachovať dvojakým spôsobom. Buď ich v CDM nebudeme uvažovať a hodnoty budeme preposielať v generickým spôsobom ako reťazce. Čím ostane úloha premapovať číselníkové hodnoty na komunikujúcich aplikáciách, rovnako ako bez použitia CDM (Typicky si konzument realizuje mapovanie na svojej strane).

Druhou možnosťou je číselníky v CDM uvažovať. Čo by mohlo byť realizované napríklad takto:

- Všetky statické výčtové typy (dni v týždni, pohlavie ...) budú definované v XSD pomocou `xsd:enumeration`. Je dôležité, aby to boli číselníky, ktoré sa nemenia, pretože prípadná zmena si vynúti

odverzovanie CDM rozhrania. (O problémoch spojených s odverzovaním CDM budem hovoriť nižšie)

- Dynamické číselníky (stavy objednávky, akademické tituly, typy dokumentov ...) by mohli byť definované napríklad pomocou SKOS, čo je W3C formát určený na vyjadrenie slovníkov alebo klasifikačných schém, ale samozrejme aj na jednoduché číselníky. Ak by sme už mali takto definované slovníky, jednotlivé XSD elementy môžeme sémanticky oanotovať pomocou atribútu `modelReference`.

Výhodou tohto prístupu by bola možnosť využitia automatického validovania a transformácie rozhrania. Použitie by sme určite našli aj počas testovania, kde by mohli testovacie nástroje čerpať svoje testovacie dáta z tohto zdroja. A ďalšou výhodou je, že s číselníkmi sa rozhranie stáva pochopiteľnejšie a lepšie sa s ním pracuje ako keď využíva generické polia.

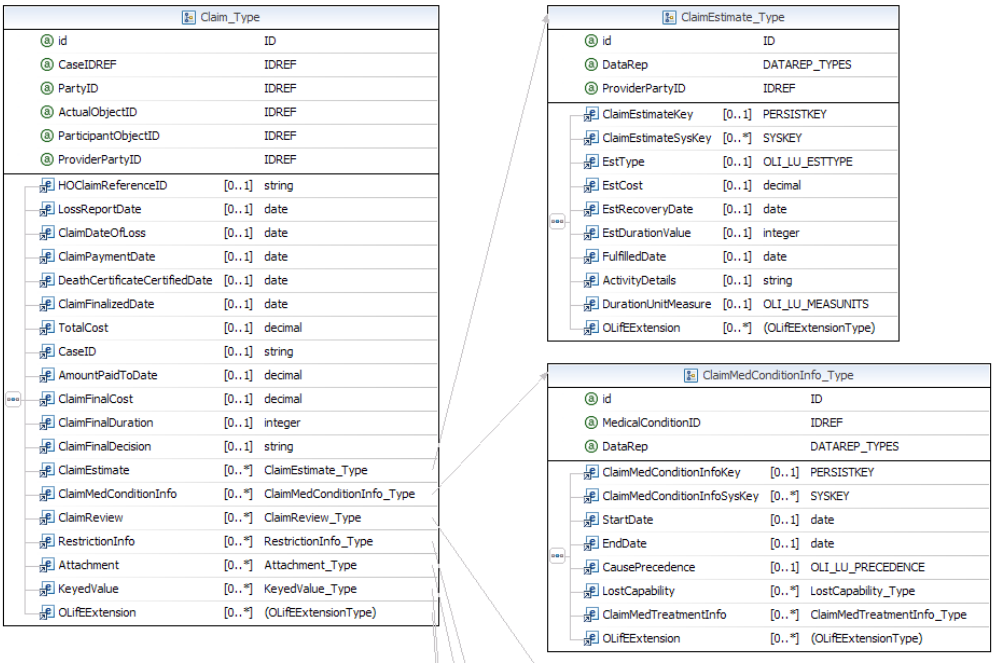
3.1.4 Štandardizované dátové modely

CDM model je možné aj zakúpiť. O zakúpení modelu je vhodné uvažovať pri vytváraní informačných systémov na zelenej lúke. Avšak v prípade spoločnosti, ktorá dlhodobo intenzívne vyvíja aplikácie svojou cestou by zanesenie nového modelu bolo príliš prácne a osobne si myslím, že aj zbytočné. Na druhú stranu ak by vznikla požiadavka B2B spolupráce medzi rôznymi organizáciami, tak by bolo potrebné dohodnúť sa na spoločnom modeli a štandardné riešenie by bol v tomto prípade asi najlepšou voľbou.

Štandardizovaný dátový model je prínosný v niekoľkých smeroch. Umožňuje využiť výsledky práce na vývoji štandardov, ktoré sa dajú počítať na človekoroky a ušetriť si tak prácu. Ďalšou výhodou štandardného dátového modelu oproti modelu vyvíjaného vlastnými silami je neutralita a objektivita a kratší čas potrebný na zaškolenie expertov. Organizácia, ktorá neprijme štandardy musí prejsť rovnakým procesom, ktorým prešli štandardizačné orgány.

Medzi najznámejšie štandardy patrí:

- SID je príkladom úspešného dátového štandardu. SID (Shared Information and Data) sa presadil v odvetví telekomunikácií a poskytovania služieb. Na vývoji a špecifikácií sa podieľajú členovia TeleManagement Forum, ktorý zadefinovali aj rámec (framework) NGOSS pre podporu prispôsobenia a rozšírenia modelu potrebám jednotlivých operátorov. [17]
- ACORD je nadnárodná, nezisková organizácia, ktorá sa od roku 1970 zameriava na vytváranie štandardov pre poisťovníctvo a súvisiace finančné odvetvia. Združuje stovky poisťovní, maklérskych firiem, softvérových domov a poisťovníckych asociácií. Títo členovia sa podieľajú na vývoji štandardizovaných formulárov a hlavne štandardizovaného výmenného XML formátu dát (poistenie majetku a nehôd, poistenie života a zdravia a prepoistenie) [12]
- HL7, HIPAA – zdravotníctvo
- ISO20022 – výmena finančných dát
- PPDM, MMDM – energetika
- FPML, SWIFT – kapitálové trhy
- CIM – (verejné) služby



Obrázok 3.2: Ukážka modelu ACORD

3.2 Transformácia

Významnou súčasťou nasadenia CDM sú transformácie. V tejto časti priblížim aké transformácie pripadajú do úvahy, ako sú technicky realizovateľné a kde sa môžu nachádzať.

3.2.1 Aké transformácie poznáme?

V zásade rozlišujeme dva druhy transformácií:

- Formátovú transformáciu
- Sémanticku transformáciu

Pod formátovou transformáciou budeme rozumieť obojsmernú transformáciu z jednej dátovej reprezentácie do druhej. Za jednoduché príklady formátovej transformácie môžeme považovať transformácie dátumu, kurzové transformácie alebo formátové transformácie peňažných súm. Patria tu ale aj číselníkové transformácie – teda transformácie z jednej sady číselníkových hodnôt do druhej.

Sémantická transformácia je zložitejší princíp. V podstate sa jedná o prevedenie jednoduchej logiky nad dátovými reprezentáciami. Napríklad: Kanonický formát rozumie entitu „poistná zmluva“, tak že obsahuje kolekciu poistených rizík a poistnú čiastku za jednotlivé riziká. Druhý systém ju chápe trochu ináč, tomu stačí o poistnej zmluve vedieť jej identifikátor a celkovú poistnú čiastku. Transformačná služba preto sčíta sumy za všetky rizika a uloží výslednú sumu. Niekto môže namietať, že ide v podstate o formátovú transformáciu, keďže sa zmenila iba dátová reprezentácia, ale uvedomme si, že kanonický formát obsahuje viac informácií, takže význam týchto entít je určite rozdielny. Takáto transformácia je pochopiteľne jednosmerná.

Je dobré aby transformačná služba vždy existovala. Dokonca aj keby lokálny doménový model bol identický s kanonickým. Takáto „nulová“ transformácia ponecháva mechanizmus generickým a umožňuje promptnejšie reagovať v prípade zmeny. Ide v podstate o všeobecné pravidlo softvérového vývoja, a to zamedzovať preskakovaniu navrhnutých vrstiev riešenia.

3.2.2 Kde transformovať?

Uvažujme, že integračné riešenie pozostáva z týchto prvkov: producent, konzument a doporučená medzivrstva middlewaru, v podobe integračného brokera (napríklad zbernica podnikových služieb). Teoreticky je možné transformovať na ktoromkoľvek z týchto prvkov, nebránia tomu žiadne technické prekážky. Dokonca je možné pre účely transformácie vytvoriť samostatnú transformačnú komponentu, ktorej jedinou úlohou by bolo transformovať poslané správy do požadovaného formátu. Technicky by mohla byť realizovaná ako EJB alebo webová služba.

Osobne by som vznik takejto transformačnej komponenty nepodporil z výkonnostných dôvodov. Každé (už tak pomalé) volanie webovej služby, by sa spomalilo ešte o ďalšie volanie transformačnej služby. Podľa Martina Fowlera základným pravidlom použitia vzdialených volaní v architektúre je minimalizovať ich počet. [7]

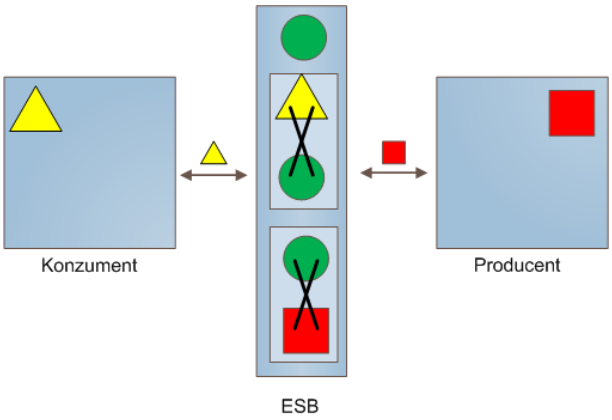
Rozhodnutie o tom, kde sa transformácia bude vykonávať je podmienené rozhodnutím o tom, v akom formáte bude služba na integračnom brokeri vystavená a či pristúpime na možnosť distribúcie CDM na klientov.

V akom formáte služby vystavovať na ESB?

Teoreticky je možné službu vystaviť na ESB v akomkoľvek formáte. Teda vo formáte CDM, producenta alebo dokonca aj vo formáte konzumenta.

Ak vystavíme službu vo formáte producenta, tak CDM sa dá využiť na transformáciu z formátu konzumenta do formátu producenta. Čo by mohlo byť realizované buď priamo u konzumenta ako adaptér[1] – podmienené distribúciou CDM. Prípadne pomocou transformačnej komponenty.

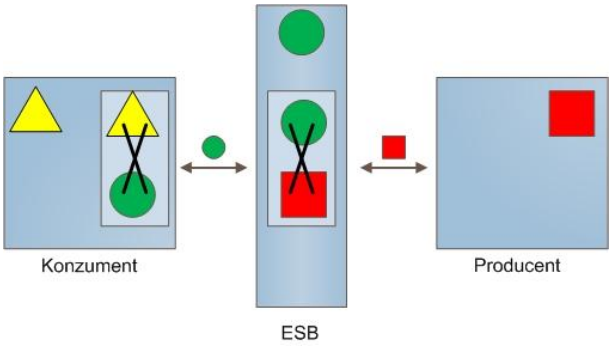
Ďalšou možnosťou je vystavovať službu vo formáte konzumenta. Takéto riešenie poskytuje maximálny komfort pre konzumenta, ktorý sa nemusí zaoberať transformáciami. Tento prístup taktiež umožňuje centralizáciu transformácií. Zmena formátu producenta alebo CDM formátu nemá na konzumentov žiaden dopad. Transformácie sú ovšem závislé na klientských formátoch. Tzn. pri zmene formátu klienta je potrebné zmeniť aj príslušné transformácie. A s nasadením novej verzie klienta, je potrebné nasadiť aj nové verzie transformácií.



Obrázok 3.3: Vystavené rozhranie vo formáte klienta

[1] Jedna sa o návrhový vzor, ktorý transformuje nekompatibilné rozhranie do kompatibilnej podoby. Inými slovami adaptér prekladá volania jeho rozhrania do volaní pôvodného rozhrania. Keďže sa jedná o medzivrstvu pri volaní externého rozhrania podporuje voľnú väzbu medzi systémami. Ako typický príklad môžem uviesť , že pri volaní externej služby producenta adaptér pretransformuje dáta z formátu klienta do formátu vystavenej služby.

Poslednou možnosťou je vystaviť službu vo formáte CDM. Konzumenti sú odtienení od zmien na producentovi. Pri nekompatibilnej zmene CDM sú však nútení sa zmene prispôsobiť. Táto možnosť je preferovaná aj z pohľadu SOA princípov, keďže služby tak budú mať voľnejšiu väzbu na producentov.

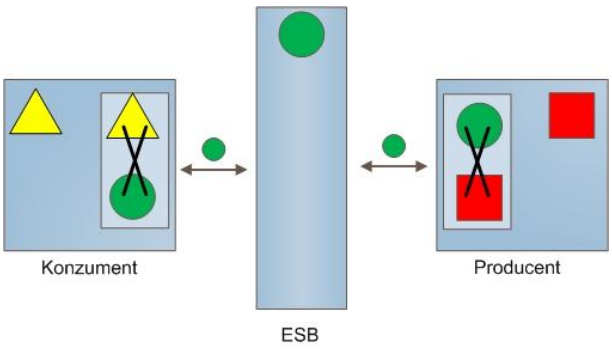


Obrázok 3.4: Vystavené rozhranie v CDM

Distribuovať CDM?

Otázka znie, či je rozumné distribuovať CDM na koncové body integračnej cesty, tzn. na producenta a klienta?

Ako by to vyzeralo keby sa CDM distribuovalo? Napríklad, tak že aplikáciám, ktoré sa chcú zúčastniť integračného riešenia by sa distribuovala knižnica s CDM objektmi. A v prípade, že by sa klient chcel zúčastniť integračného riešenia, používal by objekty CDM. To znamená ako producent by vystavil svoje rozhranie v CDM objektoch. A udržiaval by si mapovanie CDM objektov na vlastné doménové objekty. Na middlewary by sa služba vystavila v aktuálnej verzii CDM, správy by nebolo potrebné transformovať. Úlohou middlewaru by bolo iba presunutie správy medzi aplikáciami. Klient by sa pripájal na CDM rozhranie a sám by si udržiaval mapovanie medzi svojimi doménovými objektmi a CDM objektmi.



Obrázok 3.5: Príklad riešenia distribuovaného CDM

Takéto riešenie prináša niekoľko výhod:

- ESB je skutočne len poštár
- Zmena na koncových bodoch si nevynúti súčinnosť s middlewarom
- Znovupoužiteľnosť mapovania. Ak sa z producenta stane klient.
- Testovateľnosť

Veľkým problémom v tomto prípade je zmena CDM. Vzhľadom k necentrálnemu charakteru riešenia sa môžu jednotlivé závislosti stať veľmi spletitými.

Takéto riešenie by malo význam v prípade menšieho počtu kooperujúcich služieb, kde by boli dopady prípadných zmien jasné a zvládnuteľné.

3.2.3 Ako transformovat?

Ponúka sa niekoľko technických možností ako transformácie v našom prípade realizovať:

- XML na XML
- Objekt na XML
- Objekt na Objekt

Transformácia XML na XML

XML transformačné jazyky sú špecializované počítačové jazyky, určené na transformovanie vstupného XML na výstupný XML dokument. Medzi takéto jazyky patrí XSLT, XQuery, STX ...

Transformácia Objekt na XML

Existuje niekoľko nástrojov, ktoré umožňujú serializovať objekt do XML formátu. Tieto nástroje zároveň podporujú tzv. mapovací súbor, ktorý popisuje mapovanie objektu na XML a späť. Pomocou tohto mapovacieho súboru je potom možné realizovať jednoduché transformácie. Medzi tieto nástroje patrí JAXB, JiBX, Toplink, XMLBeans ...

Transformácia Objekt na Objekt.

Ďalšou možnosťou je transformovať objekt na objekt. Pre tieto účely by mohol byť použitý nástroj dozer, ktorý umožňuje deklarovať mapovací súbor popisujúci, ktoré polia sa majú na seba namapovať a následne mapovanie zrealizuje.

3.3 Riadenie zmien a dodávok

Ku zmenám CDM bude určite dochádzať, či už to budú zmeny vynútené zákonom, zmeny obchodných podmienok firmy alebo technické dôvody. Je preto potrebné byť na túto situáciu pripravený a rozmyslieť si postup nasadenia najmä nekompatibilných zmien.

Aj po úpravách CDM musí byť zachovaná konzistentnosť modelu, čo by bolo možné dosiahnuť, tak že každú požadovanú zmenu CDM musí odsúhlasiť kompetentný analytik. Pri vytváraní modelu sa musí myslieť na to, aby bol dostatočne robustný, tzn. aby dopady zmien boli minimálne.

Treba si taktiež uvedomiť, že v prostredí veľkých spoločností je striktný harmonogram postupného nasadenia aplikácií. Nie je preto možné očakávať, že všetky aplikácie budú v jeden moment používať rovnakú verziu CDM. Čo logicky ústí v možnosť potreby udržiavať rozhranie vo viacerých verziách.

3.3.1 Verzovanie webových služieb

Verzovanie API je obecný problém pri návrhu akéhokoľvek distribuovaného systému a webové služby nie sú výnimkou. Zásadným problémom je, že verzovanie nie je zabudované do architektúry webových služieb. V tejto

podkapitole preto prediskutujem niekoľko možných prístupov a doporučení , ktorými by sa dala táto sitácia riešiť.

Nekompatibilné zmeny na rozhraní sú zmeny, ktoré spôsobia, že prichádzajúca správa vyvolá chybu alebo výstupom bude nekorektný výsledok, aj napriek tomu, že pred zmenou táto správa bola spracovaná s korektným výsledkom.

Kompatibilné zmeny	Nekompatibilné zmeny
Zavedenie novej WSDL operácie	Odstránenie WSDL operácie
Pridanie nového portu	Zmena portu
Pridanie bindovania	Zmena bindovania
Zmena XSD elementu z povinného na nepovinný	Pridanie nového povinného elementu
Pridanie globálneho XSD element alebo typu	Odstránenie alebo premenovanie globálneho typu alebo elementu
Pridanie nepovinného XSD element k existujúcemu typu	Zmena elementu na nový typ, ktorý je odvodený z pôvodného odstránením nepovinných elementov
	Zmena štruktúry komplexného dátového typu
	Zmena signatúry operácie
	Odstránenie hodnoty výčtového typu
	Zmena parametrov operácie (zmena dátového typu alebo poradia)

Tabuľka 3.1: Kompatibilné a nekompatibilné zmeny

Uvažujme novú verziu rozhrania, ktoré môže byť bez zásahu použité aj klientami, ktorí boli vyvíjaní oproti starej verzii. Takéto nové rozhranie budeme považovať za spätne kompatibilné (backward compatible). Spätne kompatibilné rozhrania sú často označované jednoducho aj ako „kompatibilné“. Je možné ich dosiahnuť aplikovaním výlučne kompatibilných zmien.

Rozhranie, ktoré je navrhované tak, aby sa v budúcnosti umožňovalo napojiť aj zatiaľ neznámym konzumentom, budeme považovať za dopredne kompatibilné (forward compatible). Toho je možné dosiahnuť možnosťami XSD a to zástupnými elementmi. Takýto typ kompatibility je možné nájsť najmä v B2B scenároch. V podnikovom prostredí je najbežnejším prípadom spätná kompatibilita.

Samozrejme spätne nekompatibilné zmeny ovplyvňujú klientov, ktorý priamo volajú zmenené operácie. Na ostatných klientov nemajú vplyv.

Ak boli vykonané len spätne kompatibilné zmeny, stačí nahradiť pôvodný WSDL novým a zmeniť samotnú webovú službu. Samozrejme pôvodné WSDL zaarchivujeme, najlepšie pomocou niektorého z verzovacích nástrojov, pre prípadný návrat k staršej verzii, alebo na pohodlnejšie sledovanie zmien.

V prípade nekompatibilných zmien je potrebné službu odverzovať, tak aby z WSDL bolo zrejmé o akú verziu sa jedná. Odverzovaná webová služba by následne bola prístupná na odlišnej adrese.

V momente, keď sa takýmto spôsobom odverzuje webová služba, musíme sa rozhodnúť čo s klientmi volajúcimi starú verziu. Jedným z postojov je nerobiť nič a tým pádom prinútiť klientov sa prispôsobiť novým podmienkam, ináč volanie služby vždy skončí s chybou. Druhým zo spôsobov je ponechať starú verziu bežiacu paralelne s novou a zaviesť mechanizmus smerovania správy na základe použitého xsd dokumentu.

To, ktorý z prístupov je vhodnejší, záleží na okolnostiach. V podstate odverzovaním webovej služby nám vznikne nová webová služba, ak by sme sa rozhodli udržiavať funkčné všetky historické verzie, stálo by to viac úsilia a nákladov. Takže je žiadúce minimalizovať počet paralérne bežiacich historických verzií.

Na druhú stranu ukončenie života webovej služby znamená na strane pripojených klientov nahradenie poskytovanej funkčnosti. Čo môže byť úloha jednoduchá alebo naopak príliš zložitá. Je preto potrebné zvážiť, čo je v danú chvíľu pre spoločnosť výhodnejšie – náklady spojené s udržiavaním viacerých webových služieb alebo náklady kvôli zmene klientov.

3.3.2 Spôsob verzovania

Mechanizmus verzovanie pozostáva z dvoch základných krokov:

- Vydefinovanie artefaktov rozhrania, ktoré budú verzované.

- Vydefinovanie sady pravidiel o verzovaní, ktoré budú všeobecne uplatňované producentmi aj konzumentmi.

Medzi artefakty, ktoré je potrebné verzovať patriaa:

- XSD schémy, ktoré sa používajú na popis správ
- WSDL popis služby

Následne je potrebné dodržiavať tieto základné body:

- Pri nekompatibilných zmenách zvýšiť "Major" verziu
- Pri kompatibilných zmenách zvýšiť "Minor" verziu
- Pri zmenách netýkajúcich sa rozhrania zvýšiť "Point" verziu

"Point" verzie by mali byť použité na označenie verzii, ktoré sa týkali zmien implementácie, nasadenia alebo konfigurácie (napríklad oprava chyby, prechod na novú verziu ESB a podobne) Z tejto definície by mali byť „point" verzie kompatibilné.

Artefakt	Zmena	Major	Minor	Point
WSDL	Nekompatibilná	X		
WSDL	Kompatibilná		X	
XSD schéma	Nekompatibilná	X		
XSD schéma	Kompatibilná		X	
Kód	Oprava chýb / údržba			X
Kód	Bezpečné úpravy		X	
Kód	Logické / nebezpečné úpravy	X		
Ostatné	Ostatné			X

Tabuľka 3.2: Zmena jednotlivých typov verzií

3.3.3 Verzovanie XML schémat

Otázkou ostáva ako si v XML schéme udržiavať informáciu o verzii. Máme niekoľko možností:

Prvou z možností je využiť XML Schema atribút *version*. Problém takého riešenia je, že XML dokumenty nemajú informáciu o verzii, voči ktorej sú

validné. Nie je teda možné verziu týmto spôsobom priamo kontrolovať.Ďalšou možnosťou je použiť vlastný verzovací atribút deklarovaný pre koreňový element.

```
<xsd:schema version="2008/10/30">
  <element name="partner" type="Partner">
</xsd:schema>
```

XML dokumenty musia mať v koreňovom elemente i správnu verziu XML schémy. Výhodou je, že ak ju nemá, hneď sa to pri validácií zistí.

```
<xsd:schema>
  <xsd:element  name=" partner " type=" Partner"/>
  <xsd:complexType name="Partner">
    <xsd:attribute name="major" type="string"  fixed="12"/>
  ...
  </xsd:complexType>
</xsd:schema>
```

Výhodne je preto udržovať si takto informáciu o major verzii XSD typov. A v prípade, že major verzia XML je odlišná od major verzie XSD, XML validácia to odhalí.

```
<?xml version="1.0"?>
    <partner major="12" >
</ partner >
```

V poradí treťou možnosťou ako si uchovať záznam o verzii v XSD schéme je vloženie čísla verzie do atribútu targetNamespace. XML dokumenty môžu využívať menný priestor verzie, ktorá im vyhovuje.

```
<xsd:schema targetNamespace=".../cdm/1">
   ...
</xsd:schema>
```

```
<?xml version="1.0"?>
    <partner   xsi:schemaLocation=".../cdm/1/CDM.xsd">
</partner>
```

Ak zvolíme mennú konvenciu *targetNamespace* tak, že *targetNamespace* bude obsahovať identifikáciu XSD schémy a jej major verziu, tak dve XSD schémy budeme považovať za kompatibilné, ak je ich atribút targetNamespace rovnaký.

Je vhodné aby XML obsahovalo aj informáciu o minor a point verzii. Tieto môžu byť uložené ako atribúty koreňového elementu XSD schémy.

```
<xsd:schema>
    <xsd:element  name=" partner " type=" Partner"/>
    <xsd:complexType name="Partner">
        <xsd:attribute name="major" type="string"  fixed="12"/>
        <xsd:attribute name="minor" type="string" />
        <xsd:attribute name="point" type="string" />
        ...
    </xsd:complexType>
</xsd:schema>
```

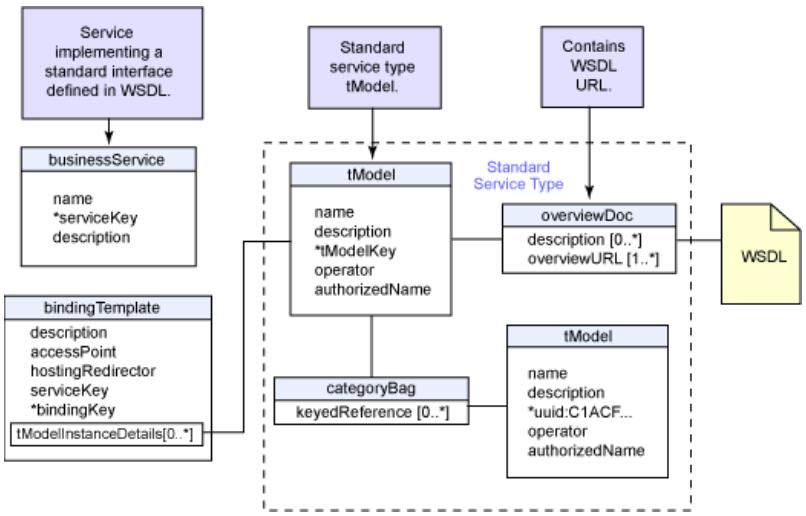
```
<?xml version="1.0"?>
<partner major="12"  minor="4"  point="1" >
</ partner >
```

Bez ohľadu na to, či je XML validácia zapnutá alebo nie, každý konzument správy musí overiť, že major verzia prichádzajúcej správy odpovedá jeho vlastnej implementácií. V prípade, že verzie nesúhlasia bude odosielateľovi vrátená informácia o chybovom stave.

Verzovanie pomocou UDDI

Verzovať služby umožňuje aj UDDI štandard. V súvislosti s UDDI sa uvažuje iba jedna verzia služby. Dátový model UDDI je však dostatočne široký nato, aby sa dal rozšíriť aj na zachytenie viacerých verzií služieb.

Riešenie je založené na predpoklade, že daná verzia `wsdl:portType` by mala byť reprezentovaná unikátnym `tModel`-om. Používatelia tejto verzie `portType` môžu vyhľadať v UDDI registry služby, ktoré sú asociované s príslušným `tModelom`, čiže sú kompatibilné s podmienkami, ktoré popisuje. V takom prípade, výsledkom tohto vyhľadávania nemôžu byť predošlé verzie, ktoré odpovedajú inému `tModelu`.



Obrázok 3.6: Dátový model UDDI [2]

3.4 Nástroje pre prácu s CDM.

Na vytvorenie a udržiavanie CDM je potrebný nástroj, ktorý by nám umožnil modelovať vzťahy medzi entitami. Výstupom má byť kanonické schéma v XML Schema jazyku. Výhodou je aj keď nástroj podporuje tímovú spoluprácu a metódy spätného inžinierstva (reverse engineering). Medzi takéto nástroje patrí Sybase PowerDesigner, Enterprise Architekt, či IBM Rational Rose Modeler.

3.4.1 Generovanie XSD z modelu

Jednou zo zásadných funkcionalít vybraného nástroja je generovanie XSD schémy z modelu. Je potrebné si rozmyslieť ako budeme pristupovať k dedičnosti a či využijeme možnosti rozšíriteľnosti XSD.

Dedičnosť

Otázkou je ako generovať dedičnosť v objektovom modeli do výsledného XSD. Možné varianty sú

- použitím abstract element a element substitution
- pomocou <choice> elementu
- pomocou abstract type a type substitution (xsi:type="[Type name]")

  Najvšeobecnejšie riešenie, ale nevedia s ním pracovať nástroje.

- pomocou kompozície

  atribúty odvodenej triedy sú realizované ako vnorený element s definovaným názvom. Takéto riešenie je preferované

Jednou z možností ako toho dosiahnuť je prispôsobiť generovanie vlastným predstavám, ak to nástroj umožňuje, prípadne preferovať už v objektovom modeli agregáciu pred dedičnosťou.

Rozšíriteľnosť

O rozšíriteľnosti XSD hovoríme vtedy, ak príslušné XML môže obsahovať dodatočné elementy, teda také elementy, ktoré XSD nešpecifikovalo. Ako je možné toho dosiahnuť?

Prvou z možností je použiť typovú substitúciu. Uvažujme nasledujúcu schému , kde element `Book` je typu `BookType`.

```
<xsd:element name="BookCatalogue">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Book" type="BookType" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Následne rozšírime typ `BookType` o nový atribút `Reviewer`, nazveme tento nový typ `BookTypePlusReviewer`.

```
<xsd:complexType name="BookTypePlusReviewer">
    <xsd:complexContent>
        <xsd:extension base="BookType" >
            <xsd:sequence>
                <xsd:element name="Reviewer" type="xsd:string"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

Výsledne XML môže obsahovať okrem atribútov `BookType` aj atribút `Reviewer`.

```
<Book xsi:type="BookTypePlusReviewer">
     <Title>My Life and Times</Title>
     <Author>Paul McCartney</Author>
```

```
    <Date>1998</Date>
    <ISBN>94303-12021-43892</ISBN>
    <Publisher>McMillin Publishing</Publisher>
    <Reviewer>Roger Costello</Reviewer>
</Book>
```

Nevýhodou takéhoto riešenia je, že rozširujúce atribúty môžu byť pridávané iba na koniec za atribúty `Book`. Ak by niekto požadoval pridať atribút na začiatok alebo do stredu, tak to týmto spôsobom nebude možné. Ďalšou nevýhodou je, že vývojár môže ľahko zabudnúť na to, že typ je možné rozšíriť a bude predpokladať iba atribúty základného typu `BookType`, čo môže viesť k chybám.

Preferovanou variantou ako riešiť problém rozšíriteľnosti je použitie elementu `xsd:any`. Rozširovanie takýchto schém je voľnejšie, autori XML majú väčšiu slobodu. Je to riešenie aj pre situáciu, že variabilita obsahu je tak veľká, že ju nie je možné postihnúť v schéme alebo ak sa štruktúra dát mení tak rýchlo, že sa nestíha schéma včas upravovať.

Na mieste elementu `xsd:any` , môže byť v XML použitý dodatočný atribút, alebo atribúty. Tu je ukážka použitia v našom type `BookType`.

```
<xsd:complexType name="BookType">
   <xsd:sequence>
       <xsd:element name="Title" type="xsd:string"/>
       <xsd:element name="Author" type="xsd:string"/>
       <xsd:element name="Date" type="xsd:gYear"/>
       <xsd:element name="ISBN" type="xsd:string"/>
       <xsd:element name="Publisher" type="xsd:string"/>
       <xsd:any namespace="##any" minOccurs="0"/>
   </xsd:sequence>
</xsd:complexType>
```

Výsledné XML môže následne vyzerať I takto:

```
<Book>
  <Title>My Life and Times
  <Author>Paul McCartney
  <Date>1998
  <ISBN>94303-12021-43892
  <Publisher>McMillin Publishing
```

```
  <rev:Reviewer>
     <rev:Name>
        <rev:First>Roger
         <rev:Last>Costello
    </rev:Name>
  </rev:Reviewer>
</Book>
```

Aby sme sa vyhli „nečakanému rozšíreniu“, teda že niekto použije odvodený typ od `BookType`, je dobré použiť `block` atribút, ktorý zakazuje použitie odvodených typov v elemente `Book`.

```
<xsd:element name="Book" type="BookType" block="#all"
             maxOccurs="unbounded"/>
```

V prípade, že používame element `xsd:any` na začiatku alebo v strede typu je dobré ho obaliť do zvláštneho elementu. Validátor totiž bez toho nevie, čo už nepatrí do sekcie `xsd:any`.

```
<xsd:complexType name="BookType">
   <xsd:sequence>
       <xsd:element name="ostatní">
         <complexType>
           <xsd:any namespace="##any" minOccurs="0"/>
         </complexType>
       </xsd:element>
       <xsd:element name="Title" type="xsd:string"/>
       <xsd:element name="Author" type="xsd:string"/>
   </xsd:sequence>
</xsd:complexType>
```

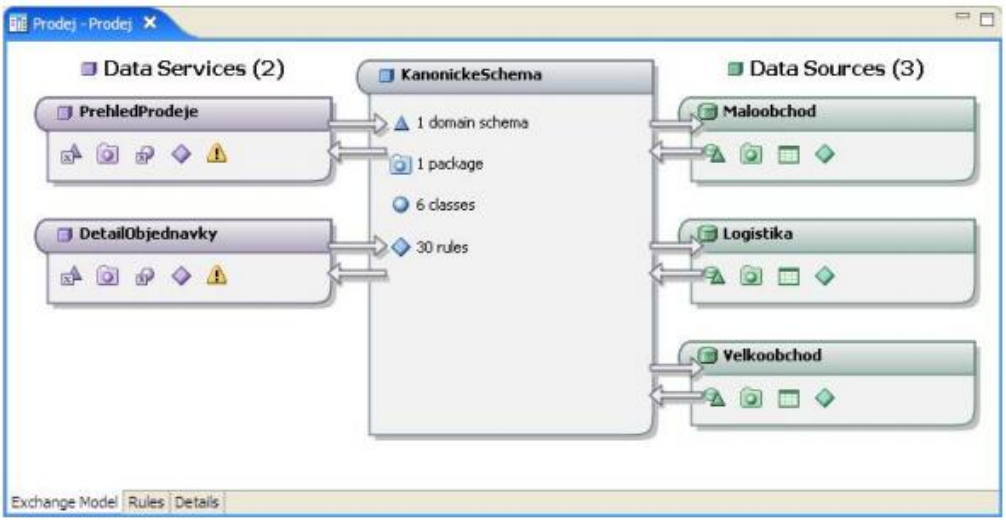
3.4.2 Progress DataXtend Semantic Integrator

Progress DataXtend Semantic Integrator umožňuje podnikovým analytikom, architektom a vývojárom vytvárať, udržovať a spravovať dátové služby pomocou spoločného (kanonického) modelu. [13] Progress DataXtend SI je následník produktu Pantero, ktorý sa orientovala predovšetkým na zákazníkov z oblasti telekomunikačného priemyslu, kde tlak na stále nové a nové služby zákazníkom vyvoláva tlak na efektívne a pružné IT. Progress sa po akvizícii úspešne snaží aj o nasadenie nástroja v iných odvetviach.

DXSI je implementovaný v prostredí Eclipse. Pre prácu s DXSI je teda vhodné poznáť základné princípy tohto prostredia. Návrhár pracujúci s DXSI by mal tiež poznať princíp návrhu diagramov tried v jazyku UML. Nutná je znalosť programovacieho jazyka Java. Znalosť XML jazykov ako je XML Schema pre špecifikáciu schémat, XSLT pre transformácie či XQuery pre dotazovanie, je možné využiť pre lepšie pochopení celkovej logiky nástroja DXSI.

Vytvorenie či import a úprava schém

Integrované komponenty, či už je to webová služba, aplikácia alebo databáza, majú definované rozhranie, skrz ktoré komunikujú s okolitými systémami. Rozhrania a spoločné modely môžu byť definované pomocou rôznych formátov; DataXtend SI umožňuje importovať tieto: XMI serializáciu UML diagramov, XML Schema, WSDL popis rozhrania webovej služby, relačnú databázu alebo Java triedy. Import z UML nástrojov uchováva vzťahy a obmedzenia špecifikované v diagrame. Tieto rozhrania importujeme do projektu ako zdroj alebo službu, tak aby platilo, že zdroj poskytuje dáta pre službu, ktorá je súčasťou obchodného procesu. Dokonca je možné rozhranie importovať ako zdroj i službu zároveň. V ďalšom kroku následne importujeme kanonické schéma. Obrázok 3.7 ukazuje výsledný integračný projektu po naimportovaní príslušných schém. Vľavo sú zobrazené dve služby, vpravo tri dátové zdroje a medzi nimi je zobrazená kanonická schéma. Šípky predstavujú transformácie, medzi jednotlivými rozhraniami. V každom smere existuje jedna, takže dve pre každú susednú dvojicu.



Obrázok 3.7: Integračný projekt v DXSI [13]

Importované schémy sú prevedené do proprietárneho DXSI modelu, ktorý je podobný UML diagramu tried a poskytuje jednotný, užívateľský prívetivý pohľad na rozhrania. Tieto je možné zároveň v tomto modeli upravovať, prípadne vytvárať. Osobne by som to však neodporúčal, pretože sa tým pripravíme o možnosť opätovného importu rozhraní. Skôr sa predpokladá, že jednotlivé rozhrania budú udržované v špecializovaných nástrojoch typu PowerDesigner, Rational Rose alebo Enterprise Architekt a po každej zmene sa rozhranie znovu naimportuje do DXSI. Tzn. ak sa rozhodneme napríklad rozšíriť rozhranie služby o nový atribút. Zanesieme túto zmenu do modelu služby a modelu CDM uloženého napríklad v PowerDesignery a zmenené rozhrania následne naimportujeme do DXSI a vytvoríme príslušnú transformáciu. Realizáciu zmien uľahčuje dopadová analýza, ktorá umožňuje vopred identifikovať, ktoré objekty a ako môžu byť zmenou ovplyvnené.

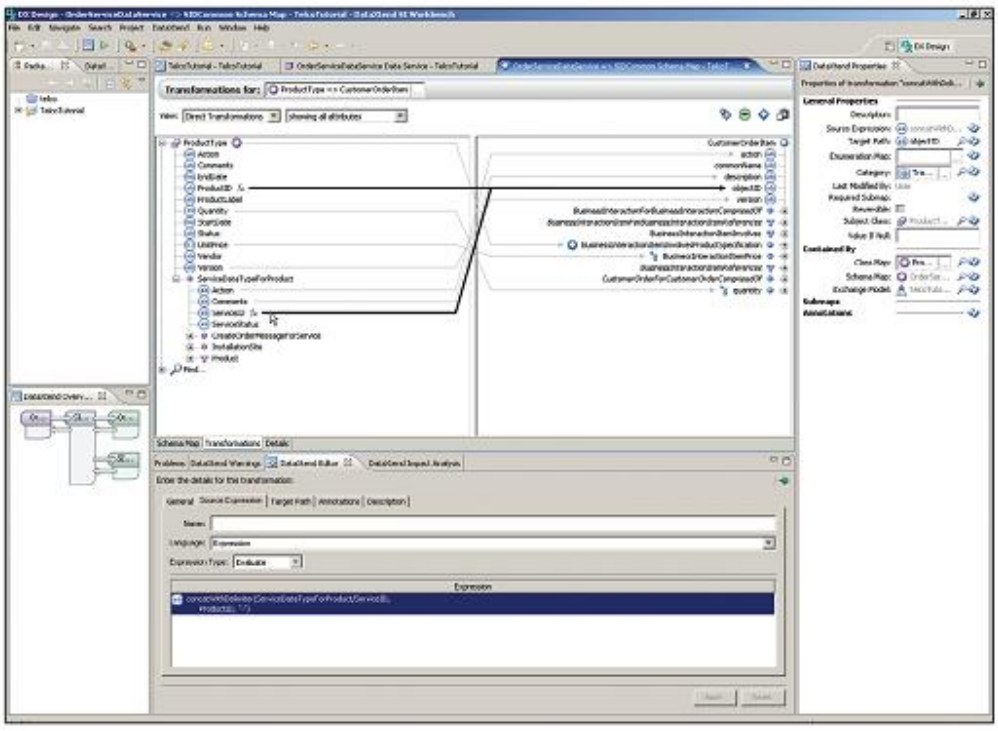
Tvorba transformácií

Kľúčovou funkcionalitou DXSI je návrh mapovaní medzi rozhraniami. Toto mapovanie definuje ako sa majú dáta transformovať, spájať a kontrolovať.

Nástroj ponúka dva jednoduché typy mapovaní:

- Mapovanie ekvivalentných tried
- Mapovanie ekvivalentných atribútov

Samotné jednoduché mapovania nemusia v reálnych projektoch stačiť. Predstavme si situáciu, že v jednom zo systémov je evidované zvlášť meno a priezvisko klienta, ale v druhom systéme sa analytici rozhodli uchovávať v jednom atribúte súčasne meno a priezvisko oddelené medzerou. DXSI rieši podobné situácie pomocou odvodených atribútov (computed attributes) a to tak, že pre každý odvodený atribút špecifikujeme výraz, pomocou ktorého sa odvodí hodnota atribútu z atribútov rovnakej alebo aj iných tried. Odvodený atribút potom mapujeme pomocou jednoduchého mapovania. V našom prípade vytvoríme odvodený atribút, ktorý naplníme zreťazením mena a priezviska oddeleného medzerou a namapujeme na cieľové rozhranie. Odvodený atribút je vlastne pomenovaná transformácia. Takúto transformáciu môžeme použiť viacnásobne a v prípade jej úpravy stačí transformáciu zmeniť na jednom mieste. Výpočet hodnoty odvodeného atribútu je možné špecifikovať ako kód v jazyku Java, výraz v jazyku XPath alebo pomocou XSDI proprietárneho jazyka, ktorý sa podoba SQL dotazom typu SELECT – FROM – WHERE. Odvodený atribút tak v sebe môže skrývať zložitý výpočet, ktorý agreguje rôzne dáta z rôznych zdrojov (FROM) na základe rôznych podmienok (WHERE) .



Obrázok 3.8: Grafické mapovanie a transformácie medzi službami

Zlučovanie dát z rôznych zdrojov je taktiež možné pomocou odvodených atribútov. V prípade, že systémy používajú rozdielne identifikátory na rovnaké entity, odvodený atribút môže obsahovať rozhodovaciu logiku naplnenia atribútov. Ďalšou funkcionalitou sú kontrolné podmienky. Jedná sa o to, že mapovanie medzi dvoma triedami môže mať na sebe naviazané kontrolnú podmienku, ktorá môže byť špecifikovaná pomocou Java kódu, XPath výrazu alebo pomocou vlastného jazyka prostredí. Táto podmienka sa vyhodnocuje pred spustením transformácie a v prípade úspechu je transformácia vykonaná. Týmto spôsobom je možné riešiť aj smerovanie dát. Napríklad podľa kódu poistnej zmluvy sa naplní odpovedajúca entita.

Testovanie

Okrem samotnej tvorby transformácií ponúka DXSI aj možnosť testovania. DXSI dokáže z popisu rozhrania vygenerovať vstupný XML súbor, ktorý môžeme použiť pre testovanie. Ak bola importovaná databáza alebo webová služba ako zdroj, je možné k nej počas testu pristupovať a získavať dáta. Ak sme však importovali ako zdroj iba UML diagram alebo java triedy, nemôže byť zdroj zahrnutý do testovania.

Po spustení testu DXSI simuluje transformáciu. Výsledkom sú XML súbory pred po transformácii do kanonického modelu ako aj po transformácii do formátu služby. Samozrejme je možné transformácie krokovať a zastaviť v ktoromkoľvek okamžiku.

Nasadenie

Po dokončení návrhu a testovania integračného projektu je potrebné nasadiť navrhnuté služby do produkčného prostredia. To DXSI umožňuje automatickým generovaním kódu pre nasadenie navrhnutých služieb v rôznych konfiguráciách. Použitie vygenerovanej komponenty pozostáva z týchto krokov:

1. Vytvoriť inštanciu služby
2. Zavolať príslušnú operáciu
3. Príjať a spracovať výsledky

Keďže inštancia DXSI služby môže byť spustená v rámci ľubovoľnej JVM (Java container), podporuje DXSI množstvo behových prostredí :

1. Java aplikační servery (JBOSS, WebSphere, WebLogic, Tomcat)

2. JMS, MQ, ESB

3. Stavová EJB

3.4.3 Open source alternatíva

Okrem špecializovaných komerčných nástrojov je možné s CDM pracovať aj pomocou open-source alternatív. Čo sa týka tvorby a udržiavania CDM modelu a modelov konkrétnych služieb môžeme použiť niektorý z dostupných UML CASE nástrojov ako je napríklad Enterprise Architect. Komerčné nástroje však navyše poskytujú zaujímavú možnosť uskutočniť dopadovú analýzu. Teda po zmene CDM umožnia zistiť, ktoré služby a ako budú ovplyvnené.

Vizuálne mapovanie objektov môžeme realizovať pomocou Altova MapForce, či Stylus Studio. Neumožnia nám však uskutočniť dopadovú analýzu zmeny CDM na mapovania, tak ako DataXtend SI. Samotné transformácie sú realizované pomocou XSLT alebo XQuery.

Testovať transformácie je možné pomocou nástrojov ako JUnit, XMLUnit alebo soapUI, biznis pravidlá je možné kontrolovať pomocou Schematronu.

Činnosť	Komerčné	Nekomerčné	Poznámka
Vytvorenie a údržba CDM modelu a modelu služieb	Power Designer, Rational Modeler	StarUML	CDM entity sú použité na popis služieb – vstupné a výstupné správy a udalosti
Popis mapovaní CDM a aplikácie	DataXtend SI, Altova MapForce, Stylus Studio	?	Slúži na komunikáciu medzi zákazníkom a vývojármi. Z tohto popisu môžeme generovať transformácie
Transformácia modelov	DataXtend SI	XSLT, XQuery	
Testovanie mapovaní počas vývoja a regresné	DataXtend SI, soapUI	Junit, XMLUnit, soapUI	Spustenie sady testov transformácií s výsledným

testovanie			reportom.
Biznis pravidlá	DataXtend SI	Schematron	
Dopadová analýza – dopad zmeny CDM na model služby	Power Designer	?	Ktoré služby události sú ovplyvnené zmenou CDM.
Dopadová analýza – mapovania z CDM do aplikačného modelu	DataXtend SI	?	Aký dopad na mapovania ma zmena CDM.

Tabuľka 3.3: Prehľad komerčných vs. nekomerčných nástrojov

3.5 Kritéria nasadenia CDM

V tejto kapitole rozoberiem, ktoré organizácie by mohli uvažovať o nasadení CDM.

3.5.1 Typ organizácií

O nasadení architektonického konceptu integrácie pomocou kanonického dátového modelu by mali uvažovať predovšetkým organizácie, ktoré majú mnoho dátových zdrojov a mnoho biznis procesov (Operating / Business System Services, OSS/BSS). V takom prípade totiž zákonite vznikajú požiadavky na častú integráciu. Samozrejme ideálne pre každú organizáciu je od začiatku vývoja systémov presadzovať využívanie kanonického modelu a nečakať kedy rôznorodosť dátových formátov začne byť neúnosným problémom.

Príklady vhodných organizácií:

- Poisťovňa sjednáva životné a neživotné poistenia do rôznych produkčných systémov. Na zjednávanie existuje niekoľko odlišných kanálov – rôznych frontendov. Každý využíva svoj špecifický dátový model.
- Telekomunikačný operátor spravuje pevné a mobilné telefónne siete a poskytuje internetové služby. Každá takáto agenda je podporovaná samostatným systémom.

- Nadnárodná spoločnosť, ktorá vznikla fúziou regionálnych organizácií, potrebuje informácie z jednotlivých národných systémov.

3.5.2 Typ integračnej architektúry

V zásade v súčasnosti existujú tieto možnosti integrácie aplikácií:

- Podniková zbernica služieb (ESB)

  Najpreferovanejší kandidát, z pozície centrálneho prestupného bodu, umožňuje centrálnu dátovú transformáciu.

- Aplikačný server

  Možnosť použiť CDM pomocou adaptérov.

- Systém riadený výmenou správ (Message Oriented Middleware)

  Taktiež možnosť použiť ako adaptéry.V prípade, že sa je potrebný vysoký výkon v reálnom čase – tisícky správ medzi malým počtom systémov bez nutnosti zložitej transformácie, je nasadenie CDM diskutabilné s ohľadom na výkonnosť (nutnosť ďalších transformácií)

- Výmenou súborov

  Teoreticky sa dá CDM taktiež použiť pomocou offline transformácie. V prípade, že by sa jednalo o XML súbory je možné použiť XSLT, v prípade iného formátu by bolo potrebné využiť iný nástroj , prípadne napísať vlastnú utilitu.

Kapitola 4

Praktická časť

V tejto časti detailne popíšem jeden konkrétny spôsob realizácie integračného riešenia použitím CDM. Nástroje pre praktickú časť boli vybrané podľa aktuálne zakúpených a používaných nástrojov v Českej pojišťovni. Samozrejme je možné použiť aj obdobné nástroje iných výrobcov (viď Tabuľka 3.3). Produkčná služba a služba v CDM formáte budú vystavené na Oracle Service Bus (OSB). A na OSB umiestnime aj XQuery a XSLT súbory transformujúce správy medzi jednotlivými formátmi (viď Obrázok 3.4). CDM model bude udržiavaný v UML diagrame pomocou CASE nástroja Power Designer 15 (PD). V tomto nástroji budú taktiež udržiavané modeli služieb v CDM formáte. Tým pádom získame možnosť dopadovej analýzy zmien CDM na vystavené rozhrania.

4.1 Vytvorenie a údržba modelu

Model bude udržovaný v PD. Ostáva ešte odpovedať na niekoľko otázok. Aký typ modelu použiť na vytvorenie CDM modelu? Ako získať výsledné WSDL rozhranie? Aké ďalšie výhody pre prácu s CDM PD prináša?

4.1.1 Sybase PowerDesigner

Je modelovací nástroj, ktorý je možné využiť na tvorbu modelov celého životného cyklu softvéru; od zberu požiadaviek, modelovanie obchodných procesov cez objektové modely až po modely dátového skladu.

Je to intuitívny nástroj s množstvom funkčností. Má dobrú podporu tímovej spolupráce v podobe perzistentného úložiska (repository). Repository umožňuje podobný štýl práce ako s verzovacími nástrojmi typu SVN alebo CVS. Umožňuje vytvoriť lokálnu pracovnú kópiu a zmeny následne porovnať oproti repository a komitovať. Samozrejmosťou je aj možnosť generovania kódu, reverse inžinieringu, či podpora tvorby reportov.

PD umožňuje po zmene na modeli uskutočniť tzv. dopadovú analýzu, čím budú odhalené všetky následky tejto zmeny. Oceníme to hlavne v prípade, že CDM je rozsiahly a webových služieb postavených nad CDM bude mnoho. V takom prípade táto analýza zobrazí všetky rozhrania, ktorých sa zmena dotkne. Na základe tohto zistenia a charaktere zmeny budeme schopný zmenu označiť za kompatibilnú resp. nekompatibilnú a podľa metodiky verzovania.

Aplikácia môže bežať ako klasická windowsovská aplikácia, prípadne ako eclipse plugin.

Kustomizácia PD

V prípade, že funkčnosť PD nie je dostatočná, PD poskytuje možnosť rozšíriť PD o požadované funkcionality.

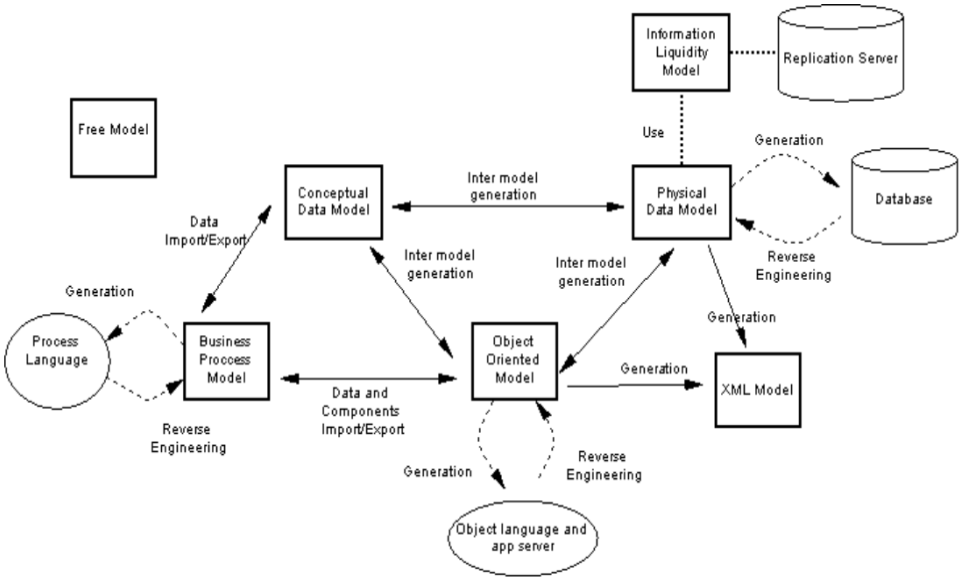
Skriptovanie

PowerDesigner ponúka možnosť písania proprietárnych skriptov, čo sa môže hodiť pri vytváraní obslužných rutín na opakujúce sa manipulácie s modelom, validácie, alebo vlastné generovanie. Taktiež je umožnené písanie obslužných rutín niektorých udalostí, transformácií a vlastných príkazov, ktoré je možné zobraziť v kontextovom menu. Na písanie týchto obslužných rutín je možné použiť Javu, C# alebo VBScript. Komunikovať s PD z inej aplikácie je možné pomocou technológie OLE.

Na druhú stranu existencia takéhoto skriptu vytvorí závislosť CDM modelu na PD a zťaží prípadný prechod na iný nástroj.

4.1.2 Výber typu modelu

Do úvahy prichádzajú tri typy modelu, objektovo - orientovaný model (OOM), konceptuálny model (KM) a XML model (XM). Aké požiadavky by mal náš model splňovať? Prvou požiadavkou je, aby sme boli schopní vygenerovať xsd popis entít. Z Obrázku 4.1 vyplýva, že z OOM je možné získať výsledné xsd, tak že sa ako generovaný jazyk zvolí XML Schema. Z OOM je taktiež možné vygenerovať priamo WSDL (ak sa do modelu zahrnie aj model webovej služby) a taktiež je z OOM možné exportovať model do xmi formátu[2]. Xsd schému je možné vygenerovať aj z XML modelu. Zároveň je z Obrázku 4.1 vidieť, že do OOM je možné pretransformovať aj konceptuálny model. Takže xsd je možné získať z OOM a XM modelu a po transformácii aj z KM.

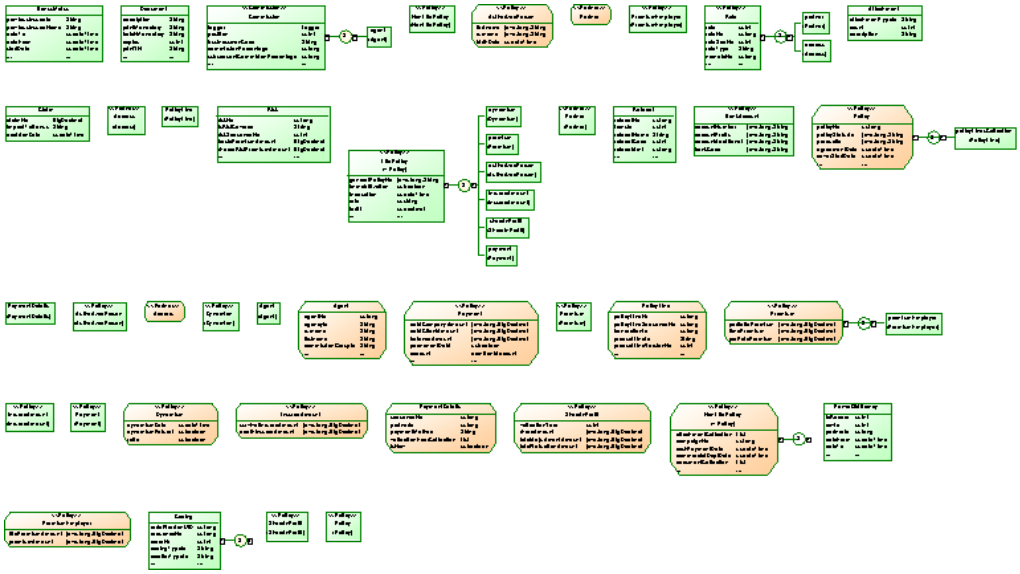


Obrázok 4.1: Transformačný graf PD modelov [15]

Ďalším kritériom pri výbere modelu je čitateľnosť modelu. Konceptuálny dátový model by mal byť najčitateľnejší aj pre netechnické role ako analytik, business analytik, zástupca užívateľov. Objektovo orientovaný model už používa technické detaily, ktoré pre netechnické role pôsobia komplikovane. XML model je najbližšie cieľovému XSD ale jeho čitateľnosť je

[2] XMI, XML Metadata Interchange je OMG štandard pre výmenu metadát pomocou XML. Najčastejším použitím XMI je výmenný formát UML, ale môže byť použitý aj pre ostatné modeli, ktorých metamodel je možné vyjadriť v MOF.

problematická aj pre technické role, keďže je neprehľadný a nie sú z neho jasné vzájomné vzťahy entít.



Obrázok 4.2: XML Model

V prípade CDM ako jednotného modelu pre integračné účely, ktorý je primárne určený pre technické role, by som preferoval udržiavať model v OOM. OOM je bližšie výslednému XSD ako konceptuálny model, čím by sa ušetrila jedna transformácia (z konceptuálneho modelu do objektového modelu), čím sa vyhneme zbytočným komplikáciám (Napríklad KDM neobsahuje zložené dátové typy).

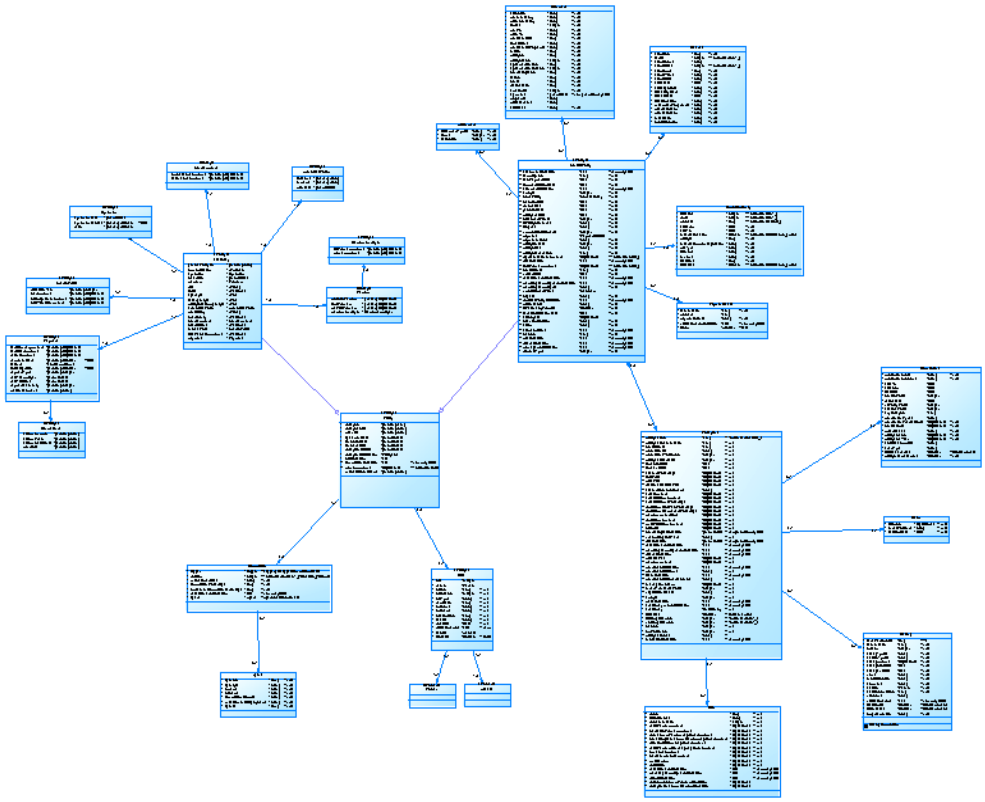
4.1.3 Vytvorenie modelu poistnej zmluvy

V tejto kapitole sa pokúsim navrhnúť kanonický formát poistnej zmluvy, čo je kľúčová entita poistného odvetvia. V súčasnom stave neexistuje v ČP jednotný pohľad na formát poistnej zmluvy. Tak ako decentralizovane vznikali jednotlivé aplikácie, vznikali aj rozdiely v použitom doménovom modeli. A v konečnom dôsledku každá aplikácia si definuje svoj vlastný formát poistnej zmluvy. Cieľom teda je navrhnúť zastrešujúcu entitu pre integračné účely, ktorá by vyhovovala všetkým aplikáciám, teda takú, ktorá obsahuje všetky integrované dáta.

V prvom rade bolo potrebné si rozmyslieť, či budem model tvoriť od začiatku, nanovo alebo použijem existujúce modeli. Vytvorenie nového modelu, by si

vyžadovalo nemalé analytické skúsenosti z poisťovníckeho odvetvia a v prípade, že by sa výsledok odlišoval od modelov existujúcich aplikácií, hrozilo by, že transformácie budú natoľko rozsiahle, žeby v konečnom dôsledku žiadna z aplikácií nebola spokojná. Preto som sa radšej rozhodol ísť cestou menšieho odporu a pri tvorbe výsledného modelu som vychádzal z existujúcich modelov dominantných systémov. V ČP existujú dve veľké produkčné systémy. Jeden určený na zjednávanie neživotných poistných zmlúv (poistenie áut, majetku, zodpovednosti …) a jeden určený na zjednávanie životných poistných zmlúv. Každý z týchto systémov používa vlastný model. Keďže sa jednotlivé agendy poistovníctva líšia, tak sa pochopiteľne líšia aj doménové modeli, ktoré používajú a to pomerne výrazne.

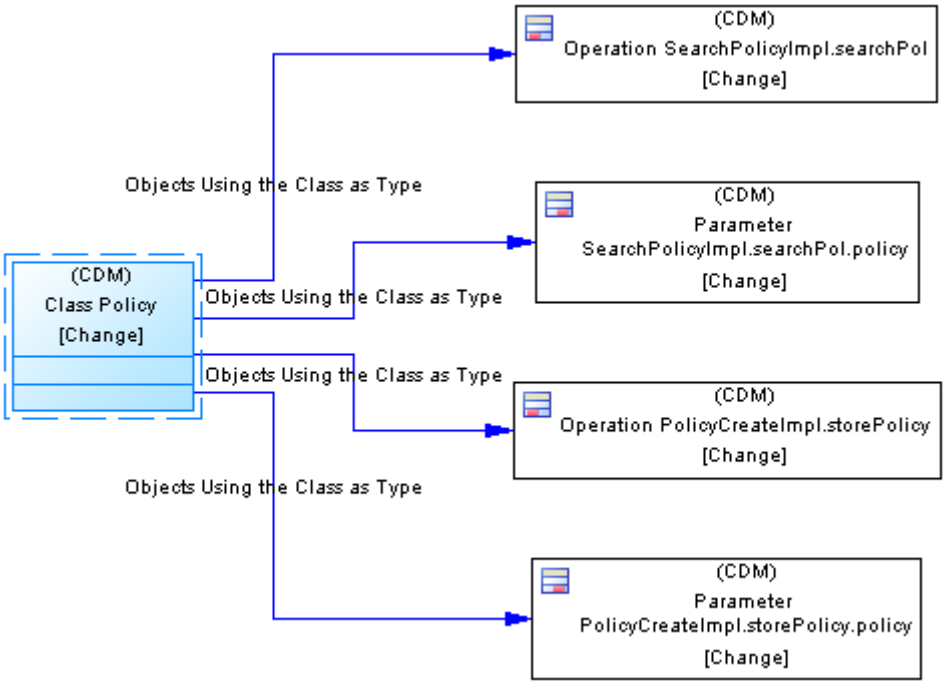
Postupoval som teda tak, že som sa snažil identifikovať čo najviac atribútov a entít, ktoré som umiestnil do spoločnej časti a špecifické informácie som umiestnil do tried pre životné a neživotné poistenie. Snažil som sa vyhnúť dedičnosti, keďže s ňou majú nástroje problémy (viď kapitolu 3.4.1 Generovanie XSD z modelu) a namiesto nej som radšej používal kompozíciu.



Obrázok 4.3: Ukážka modelu CDM poistnej zmluvy

4.1.4 Vytvorenie a údržba modelu CDM služieb

Je dobré si udržiavať modely CDM služieb v Power Designery. Parametre jednotlivých služieb budú tvorené entitami CDM modelu. Prináša to výhodu dopadovej analýzy zmien CDM modelu na rozhrania služieb. To znamená, že po zmene CDM budeme schopný rýchlo a presne určiť, ktoré rozhrania boli zmenou ovplyvnené. Dopadová analýza funguje na úrovni tried. Model služby môžeme taktiež využiť ako východiskový bod pre generáciu XSD, respektíve WSDL.



Obrázok 4.4: Diagram dopadovej analýzy

Pri generovaní modelu do WSDL je potrebné brať v úvahu menné konvencie a vnútropodnikové štandardy. Aby sme toho dosiahli budeme musieť pozmeniť spôsob generovania.

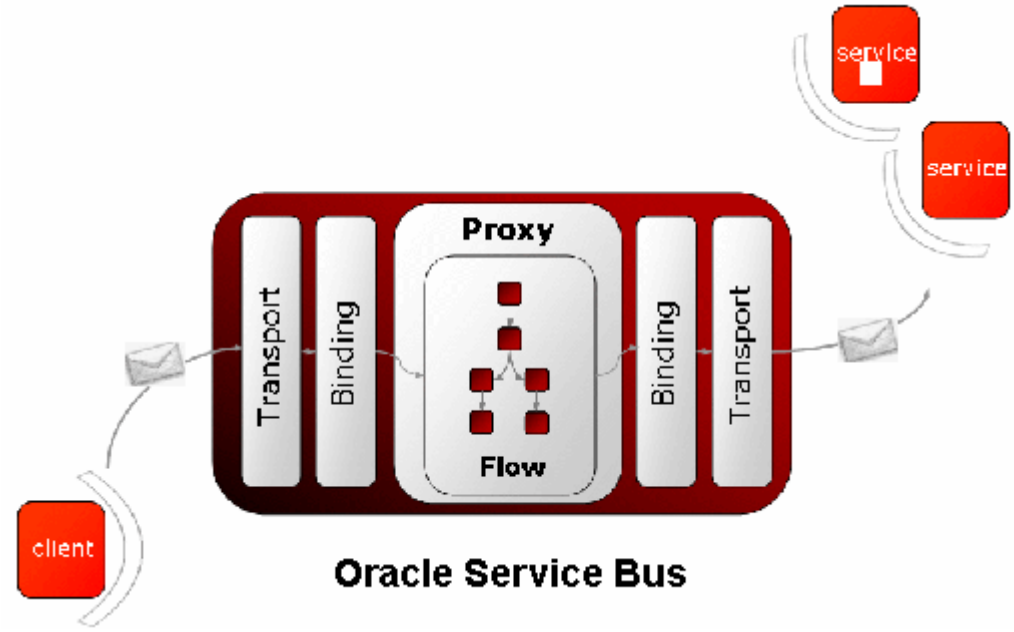
4.1.5 Vytvorenie transformácií

Model transformácií ma význam iba ak sa z neho generujú samotné transformácie. V opačnom prípade sa po čase stane model neaktuálny. Existuje niekoľko nástrojov na vizualizáciu mapovaní a vygenerovanie transformácií. Medzi ne patrí Stylus Studio, Altova MapForce. Vítanou by

bola funkčnosť dopadovej analýzy zmeny CDM na mapovania. Tú však tieto nástroje neposkytujú. Preto nepoužijem žiaden špecializovaný nástroj a transformácie napíšem pomocou dostupných XSLT, či XQuery editorov.

4.2 Nasadenie na Oracle Service Bus

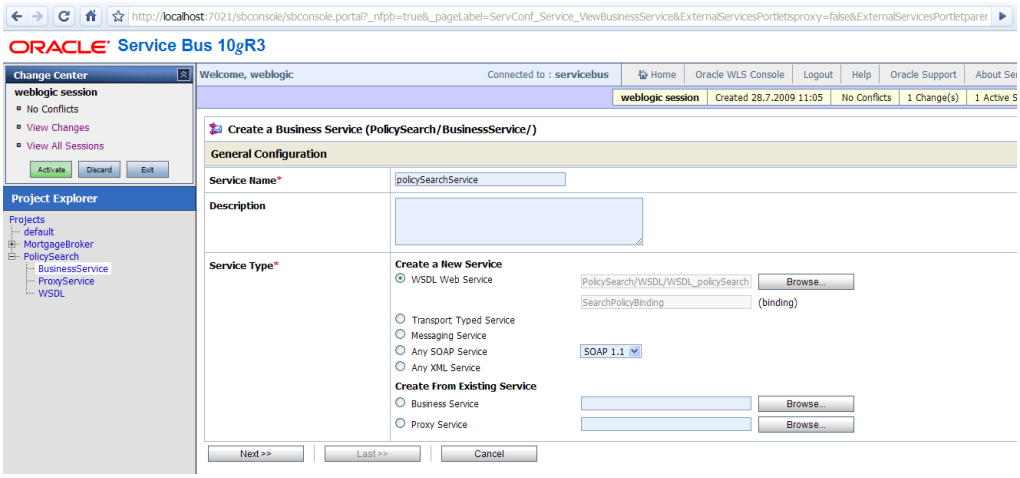
V okamžiku keď disponujeme WSDL popisom rozhrania CDM služby, službou producenta a transformáciami medzi nimi je možné pristúpiť k nasadeniu všetkých častí na OSB. V prvom rade nainštalujeme OSB a pomocou nástroja Configuration Wizard vytvoríme doménu, v ktorej si jednotlivé zdroje (resources) nasadíme. Obrázok 4.5 ilustruje fakt, že OSB volanie môžeme rozdeliť na tri časti. V prvej časti sa správa transportuje ku proxy komponente, ktorá sa postará o zavolanie služieb, ktoré správu spracujú a nakoniec sa výsledok prepošle ďalej. Buď je to ďalšia proxy služba, alebo už návratová správa určená klientovi. V našom prípade pôjde o vytvorenie proxy služby s rozhraním vo formáte CDM, ktorý sa následne pretransformuje do formátu volanej služby producenta. Všetky tieto úkony budeme nastavovať v OSB administrátorskej konzoli, ktorú môžete vidieť na Obrázku 4.8.



Obrázok 4.5: OSB architektúra [14]

4.2.1 Vytvorenie Proxy služby

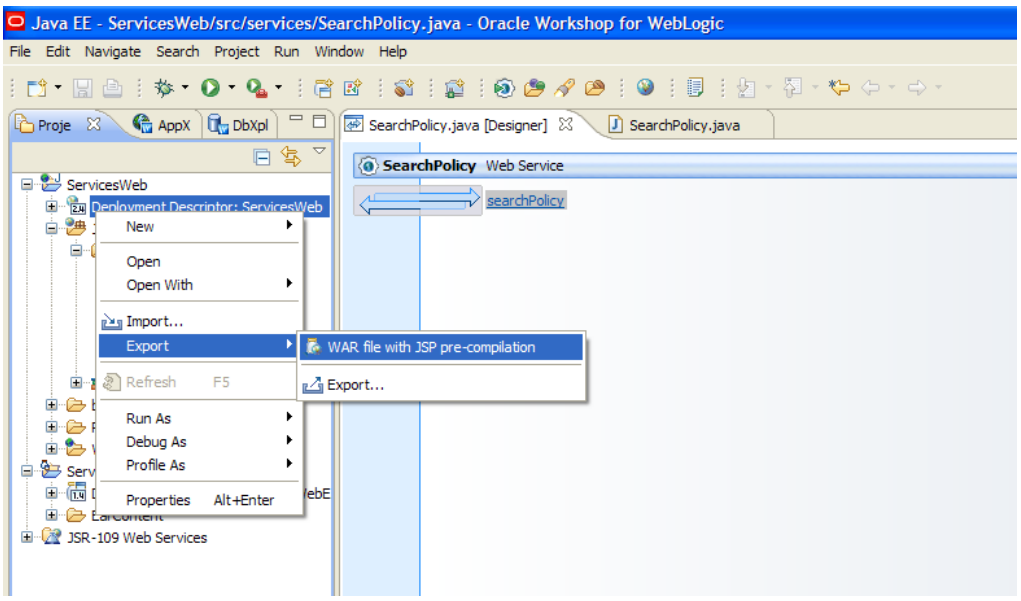
Proxy službu `searchPolicyProxy` vytvoríme tak, že naimportujeme WSDL, ktoré sme získali z Power Designera a nastavíme rôzne monitorovacie, bezpečnostné a behové parametre. Napríklad je možné určiť výsledné URI, na ktorej bude služba prístupná, alebo agregačný interval, čo je čas, za ktorý sa budú zhromažďovať a zobrazovať monitorovacie dáta. Taktiež je možné nastaviť SLA (Service Level Agreement), čo je monitorovanie dohodnutých charakteristík služby, ako napríklad čas odozvy (response time), počet správ alebo počet chýb. Porušenie týchto sledovaných pravidiel je následne reportované.



Obrázok 4.6: Naimportovanie WSDL v OSB konzole

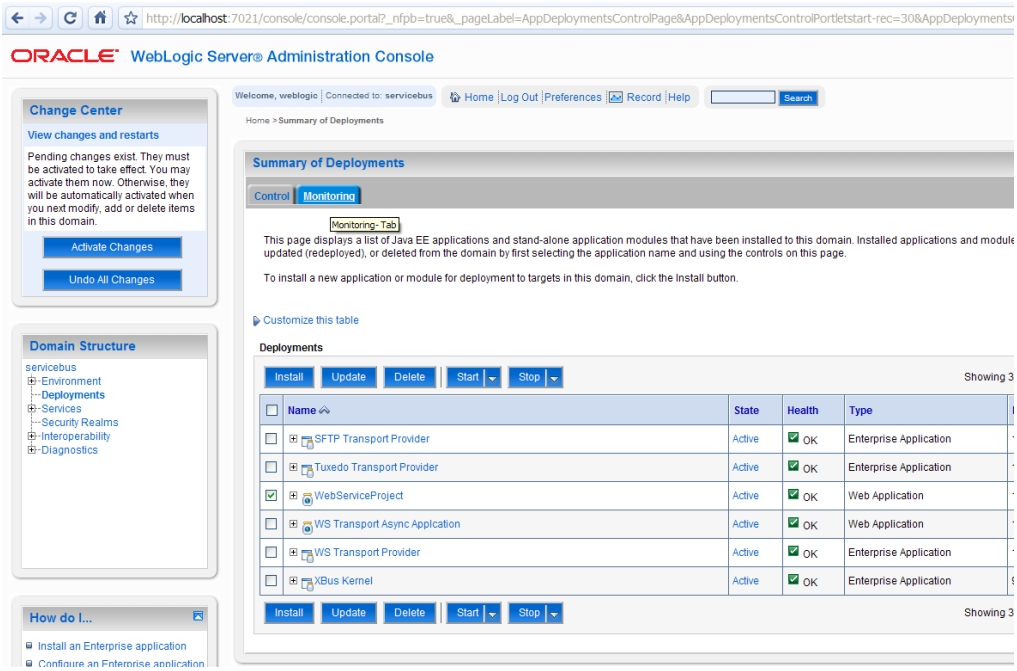
4.2.2 Vytvorenie biznis služby

To čo OSB nazýva biznis službou (business service) je v našom názvosloví služba producenta, teda už konkrétna služba ktorá spracuje požiadavku. Na vytvorenie testovacej služby bolo použité Oracle Workshop IDE. Je to nástroj vytvorený v prostredí eclipse, špecializovaný pre weblogic produkty. Práca s ním bola veľmi pohodlná a intuitívna. Z projektu webovej služby bol vytvorený war balíček, tak ako je vidieť na Obrázku 4.7.



Obrázok 4.7: Vytvorenie balíčka webovej služby

Ďalším krokom bolo nasadenie balíčku na Weblogic Server pomocou Weblogic Server administrátorskej konzoly. Je potrebné zvoliť umiestnenie war balíčku a nastaviť parametre aplikácie.

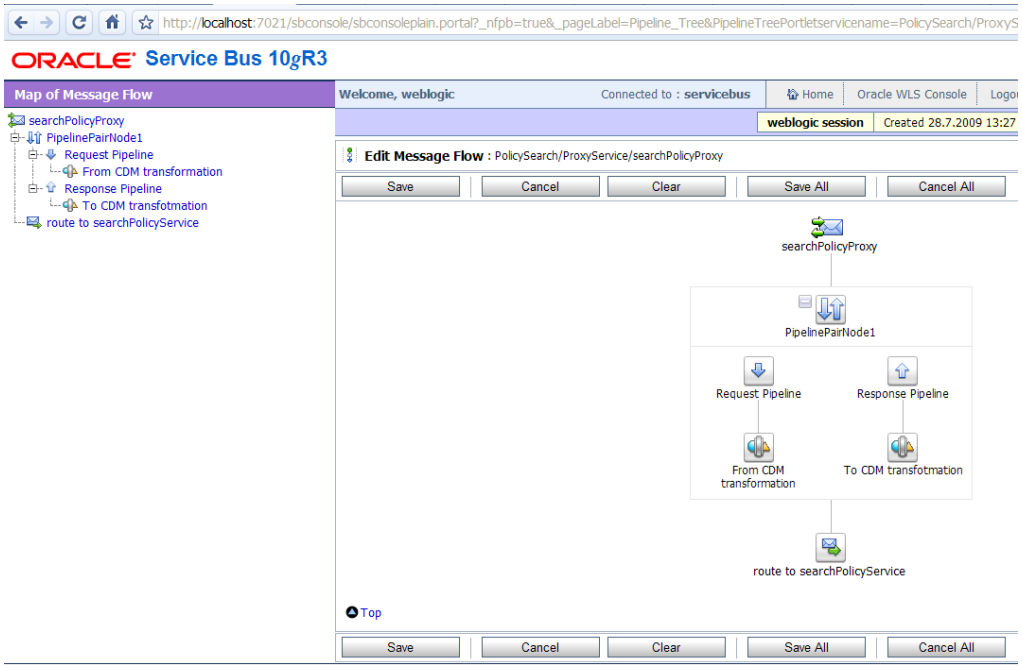


Obrázok 4.8: Weblogic Server administrátorská konzola

Po nasadení služby naimportujeme do OSB konzoly nové WSDL, ktoré následne použijeme pri vytvorení biznis služby, podobne ako sme vytvorili proxy službu.

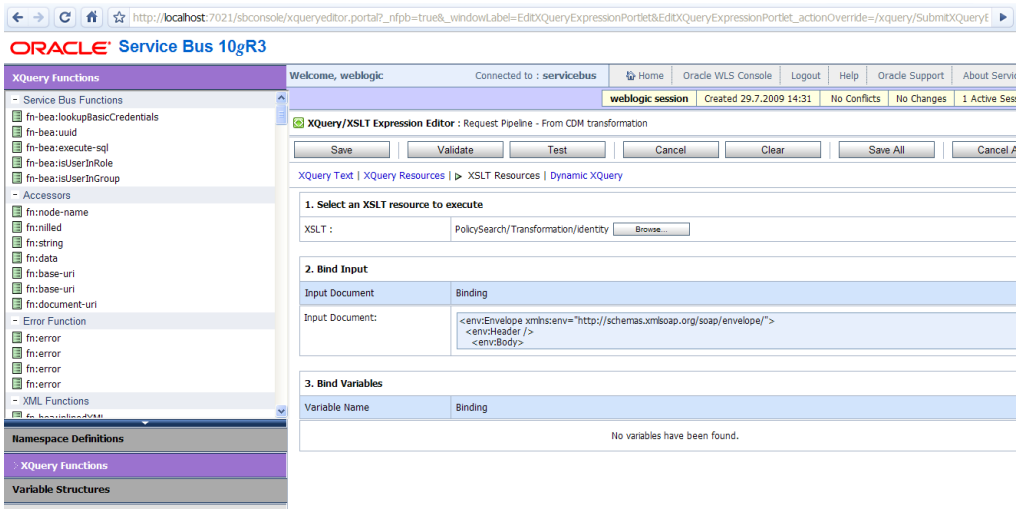
4.2.3 Nastavenie transformácií

Ďalším krokom je nastavenie volania biznis služby službou proxy. Toho je možné dosiahnuť tak, že na proxy službe zvolíme možnosť editovať tok správy. Obrázok 4.9 ukazuje, tok správy – je možné vidieť, že samotnému presmerovaniu na biznis službu `searchPolicyService` sme predradili transformácie z CDM formátu a do CDM formátu.



Obrázok 4.9: Tok správy

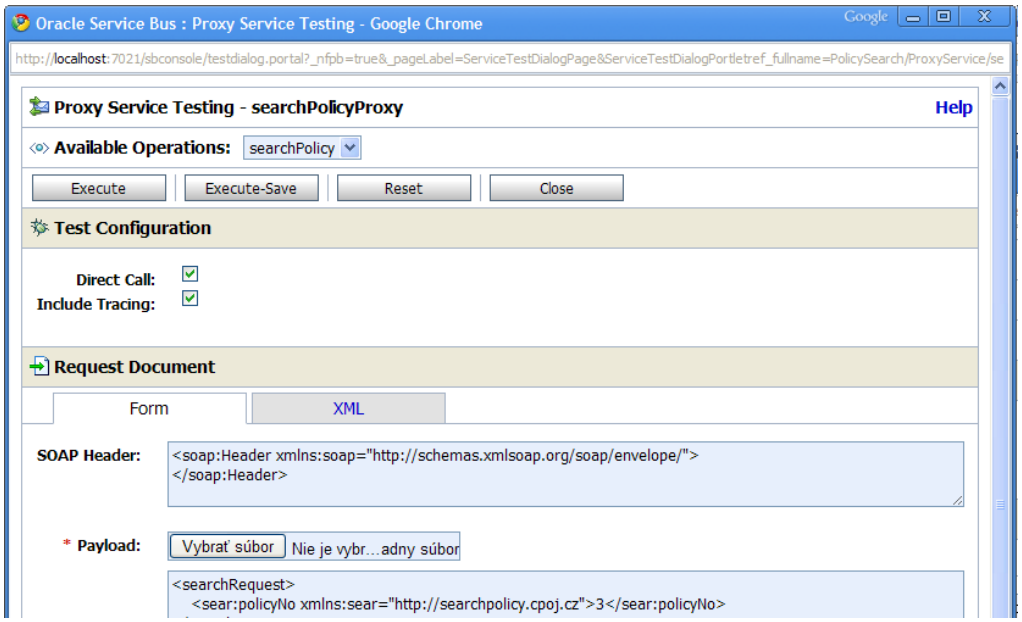
Transformácie je možné definovať pomocou XQuery alebo XSLT transformácií v XQuery/XSLT Expression Editore.



Obrázok 4.10: XQuery/XSLT Expression Editor

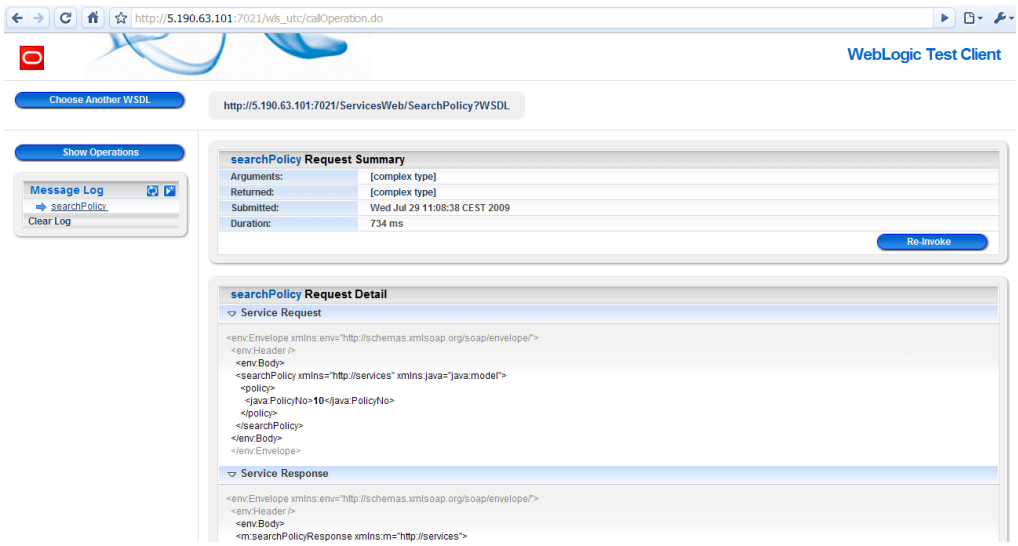
4.2.4 Podpora pre testovanie

OSB umožňuje testovať jednotlivé služby pomocou dvoch spôsobov. Prvým je zabudovaný webový klient v OSB administrátorskej konzole, ktorý umožňuje zostaviť SOAP request a vykonať ho.



Obrázok 4.11: Zabudovaná testovacia konzola služby

Druhým spôsobom je dodávaná aplikácia Weblogic testovací klient (Weblogic Test Client), ktorú je možné nájsť na adrese `/wls_ut`. Umožňuje vybrať ľubovoľné WSDL a otestovať všetky operácie, ktoré popisuje.



Obrázok 4.12: Weblogic testovací klient

4.3 Riadenie dodávok

Veľké spoločnosti majú presný harmonogram dodávok jednotlivých systémov (tzv. release plan), ktoré na seba logicky nadväzujú podľa závislostí medzi jednotlivými systémami. Otázka znie ako do tohto ťažko meniteľného harmonogramu zakomponovať zmenu CDM, respektíve zmenu služieb v CDM formáte.

Začnime jednoduchším prípadom. V prípade kompatibilnej zmeny, teda zmeny, ktorá neovplyvní klientov služby je zmenu možné nasadiť prakticky kedykoľvek. Zmeny budú realizované na jednom mieste a to na OSB. Po zmene CDM v PD si vygenerujeme príslušné WSDL a nasadíme ho na vystavenú proxy službu. Súčasne zmeníme transformácie ak to bude potrebné.

Komplikovanejší prípad je ak sa na CDM uskutočnia nekompatibilné zmeny. V takom prípade musí vzniknúť nová verzia služby. V prípade, že sa spolu so zmenou CDM rozhrania zmení (a nasadí) aj klient, tak je všetko v poriadku. Problém nastáva, keď klient nemôže prejsť na nové rozhranie. A to z dôvodu rozdielneho termínu vydania (releasu), alebo z iných dôvodov (z kapacitných - vývojový tím na to nemá čas, alebo z finančných - nikto takú zmenu nezaplatí). V takom prípade je potrebné ponechať funkčnú aj starú verziu služby. To môžeme urobiť niekoľkými spôsobmi. Vždy vznikne nová proxy služba, rozdiel je ale v tom, čo táto proxy služba volá:

- nová proxy službu volá novú biznis službu

  Vytvoríme novú proxy službu s novým CDM rozhraním, ktoré bude volať novú implementáciu služby. Súčasne ponecháme aktívnu aj starú službu, teda staré rozhranie a implementáciu. Použijeme napríklad, ak sa zmenila aj implementácia služby.

- nová proxy služba volá rovnakú biznis službu ako tá stará

  Nová proxy služba bude vystavovať nové CDM rozhranie, za ktorým budú aj nové transformácie. Obe rozhrania budú volať rovnakú implementáciu služby.

- nová proxy služba volá pôvodnú proxy službu

Nova proxy služba bude vystavovať rozhranie s novým CDM formátom, správa sa následne pretransformuje do formátu starého rozhrania a zavolá sa pôvodná proxy. Takže nová proxy služba bude vlastne slúžiť ako adaptér medzi starou a novou verziou CDM.

4.4 CDM procesy

V tejto kapitole zadefinujem procesy, ktoré sa nad CDM budú realizovať.

Procesov sa budú účastniť tieto role:

- Analytik IS
- Integrační architekt
- Správca CDM - osoba poverená za fyzické úpravy modelu

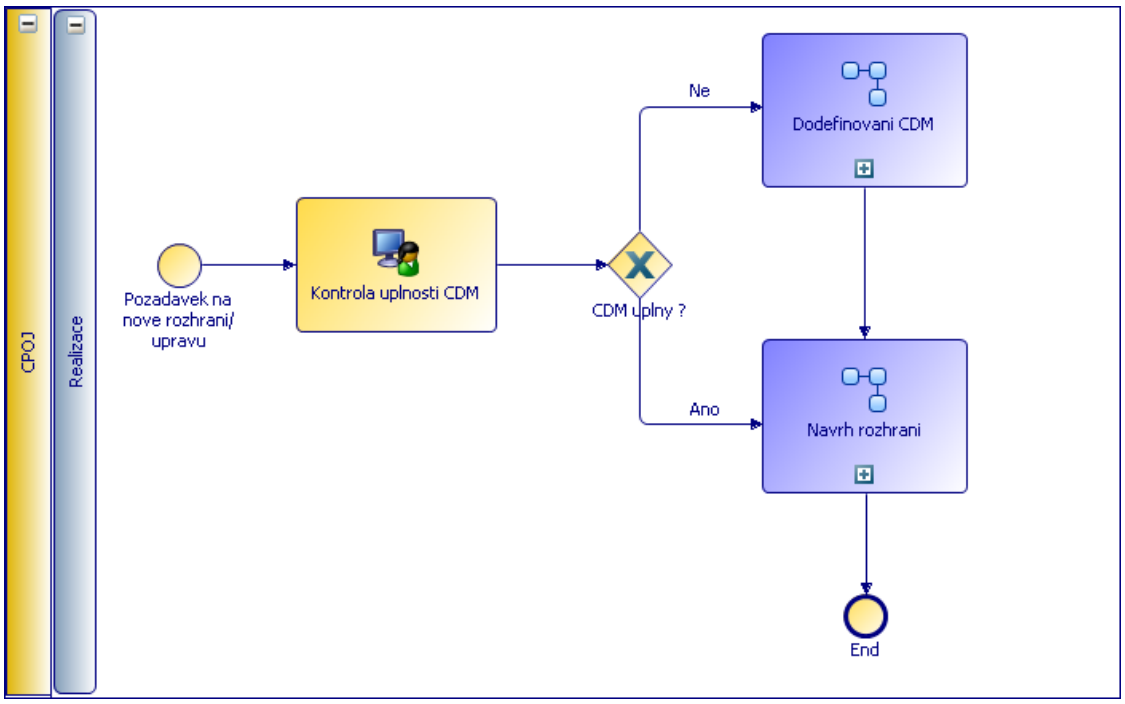
4.4.1 Požiadavka na nové rozhranie

Aktéri procesu:

Analytik požiadavku, Správca CDM

Popis procesu:

Spúšťacou udalosťou je požiadavka na nové rozhranie, prípadne na jeho úpravu. Na základe tejto požiadavky analytik požiadavky v spolupráci so správcom CDM zverifikuje, či CDM obsahuje všetky entity a atribúty potrebné pre zrealizovanie požiadavky. V prípade, že tomu tak je, nasleduje samotný návrh rozhrania. V prípade, že CDM nie je úplný tomuto kroku ešte predchádza dodefinovanie CDM.



Obrázok 4.13: Procesný diagram požiadavky na nové rozhranie

4.4.2 Požiadavka na zmenu rozhrania

Aktéri procesu:

Analytik požiadavky, Správca CDM

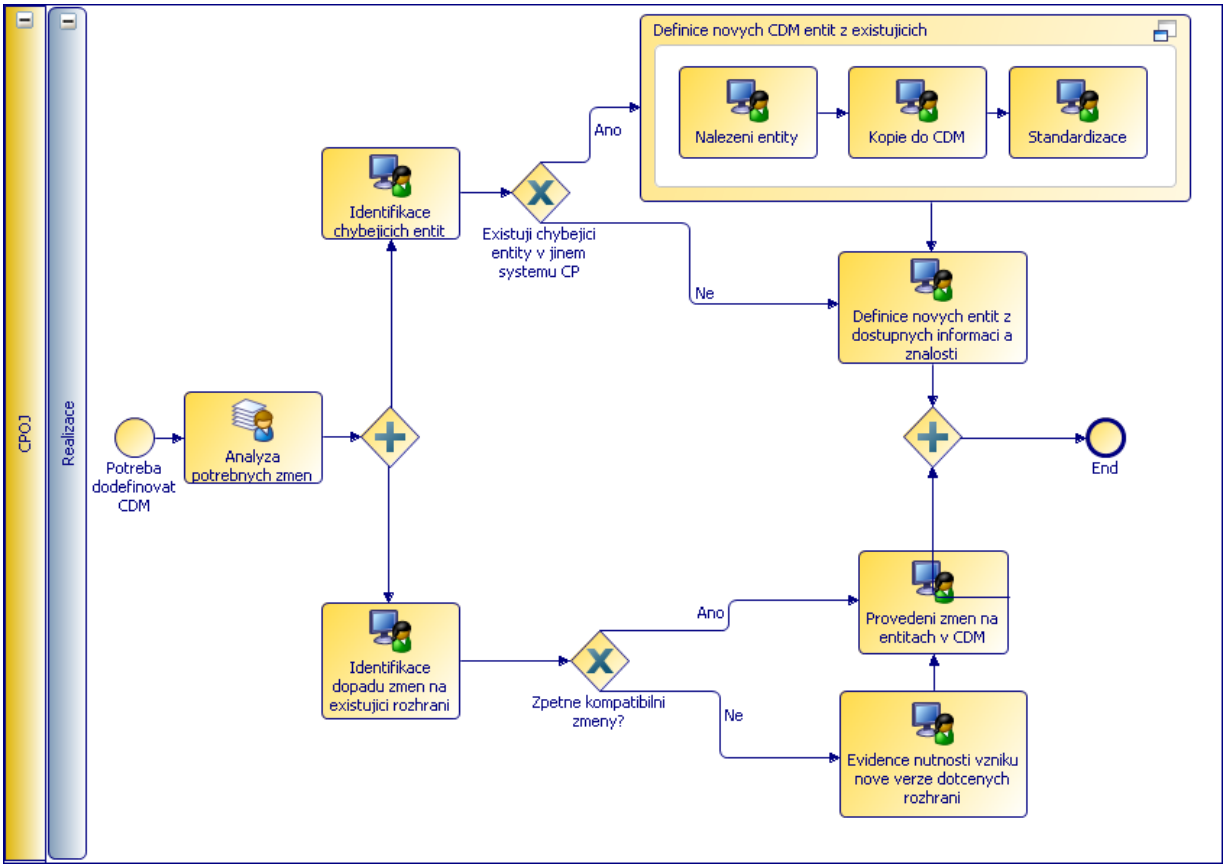
Zdroje procesu:

Sybase Pover Designer, Dátové modely systémov ČP, Externý referenčný model

Popis procesu:

Analytik požiadavky alebo solution architekt požiadavky v spolupráci so správcom CDM uskutoční analýzu požadovaných zmien. Analýza môže ukázať, že v CDM sú potrebné úpravy. V prípade, že v CDM chýba entita, pokúsime sa ju identifikovať v rámci existujúcich systémov v rámci organizácie, prípadne konzultovať s dostupným štandardným modelom. Po jej nájdení ju skonsolidujeme (prevod názvov a typov podľa integračných štandardov) a zaradíme do CDM modelu. V prípade, že sa jedna o doposiaľ neexistujúcu entitu, je potrebné ju navrhnúť podľa dostupných znalostí (ideálne externý referenční model) a integračných štandardov.

V prípade, že je potrebné upraviť existujúce entity, je nutné najprv uskutočniť analýzu dopadu (PD Impact analysis) a na základe výsledku analýzy uskutočniť požadovanú zmenu. Uskutočnené zmeny môžu byť spätne nekompatibilné (tzn. pregenerovanie rozhraní by si vyžiadalo zmenu na strane klientov rozhraní). V takom prípade je potrebné odverzovať dotknuté služby.



Obrázok 4.14: Procesný diagram požiadavku na zmenu rozhrania

Kapitola 5

Záver

V tejto práci som zanalyzoval a navrhol integračné riešenie pomocou CDM, ktoré by mohlo byť potenciálne použité v prostredí Českej poisťovne. Zhrnul som súčasné možnosti ako pracovať s CDM a poskytol som ukážkové prototypové riešenie. Poukázal som na problémy, ktoré pri zavádzaní CDM je potrebné riešiť a uvažovať. Takýto zhrňujúci dokument zatiaľ medzi verejne dostupnými zdrojmi chýbal.

Kľúčové pre úspech CDM je kvalitný model. To znamená model, na ktorom by spolupracovali skúsení analytici a ktorý by bol obecne prijateľný pre celú organizáciu. Rizikové nie sú technológie, tie sú pre prácu s CDM pripravená už aj dnes.

Tým, že som pri písaní spolupracoval s firmou, nahliadol som do procesu tvorby softvérových riešení pre veľké organizácie, čo považujem za neoceniteľnú skúsenosť.

Počas písania práce som si uvedomil, že budúci výskum by sa mal zamerať na podporu práce s mapovaním. To znamená vytvorenie priemyselného štandardu pre formálne a sémantické mapovanie, jeho súčasťou by taktiež mal byť formát pre uloženie biznis pravidiel, invariantov a ich napĺňanie v behovom prostredí. Následne by som očakával nástup nástrojov podporujúcich tieto štandardy. Taktiež prínosné by bolo rozšírenie nástrojov pre vizuálne mapovanie a dopadovej analýzy zmien mapovaných zdrojov na samotné mapovania, respektíve transformácie.

Tak ako doposiaľ nebola vynájdená technológia ani metodika, ktorá by vyriešila komplexitu softvérového vývoja [1] ani CDM nie je spasiteľným riešením systémových integrátorov. Každý nástroj a aj CDM je možné

používať nesprávne. Som však presvedčený, že CDM pri vhodnom použití môže priniesť z dlhodobého hľadiska veľkým spoločnostiam nemalé úspory. CDM taktiež zapadá do filozofie SOA o flexibilných a nezávislých službách, tým že vytvára medzi integrovanými bodmi ďalšiu vrstvu indirekcie.

Dúfam, že aj táto práca raz poslúži ako podklad pri zvažovaní o tom, či a ako je možné CDM v danom prípade použiť.

Zoznam použitej literatúry

[1] Brooks Fred, *The Mythical Man Month*, Addison-Wesley, 1995

[2] Brown Kyle, Ellis Michael, *Best practices for Web services versioning*, http://www.ibm.com/ developerworks/webservices/library/ws-version, 2004

[3] Džmuráň Michal, *Datová integrace nezbytná součást SOA*, 2008

[4] Džmuráň Michal, *Úvod do datové integrace řízené společným datovým modelem*, 2007

[5] *Európsky parlament*, http://www.europarl.europa.eu/parliament/public/staticDisplay.do?id=155&pageRank=4&language=SK, 2008

[6] Fowler Martin, *Integration Database*, http://www.martinfowler.com/bliki/IntegrationDatabase.html, 2008

[7] Fowler Martin, *Patterns of Enterprise Application Architecture*, 2002

[8] Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, E., Addison Wesley, 1995

[9] Hohpe, G., Woolf, B., *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, Addison-Wesley Professional, 2003.

[10] Howard Philip, *The importance of a common data model*, http://www.itdirector.com/technology/applications/content.php?cid=10292, 2008

[11] Jack van Hoof, *How to mediate semantics in an EDA*, http://soa-eda.blogspot.com/2007/04/how-to-mediate-semantics-in-eda.html, 2009

[12] *Insurance data standards*, www.acord.org, 2009

[13] Nečaský Martin, Štumpf Jindřich, *DataXtend SI: datová integrace řízená společným schématem*, 2008

[14] *Oracle Service Bus Architecture*, http://download.oracle.com/docs/cd/E13159_01/osb/docs10gr3/concepts/architecture_overview.html, 2009

[15] *PowerDesigner General Features Guide*, http://infocenter.sybase.com/help/topic/com.sybase.stf.powerdesigner.docs_12.5.0/pdf/GeneralFeatures.pdf, 2007

[16] Rajský Peter, *Theory and Practice of System Integration* http://it.toolbox.com/blogs/system-integration-theory, 2009

[17] *Service providers and their suppliers data standards*, www.tmforum.org, 2009

[18] *SOA Design Patterns and Pattern Languages*, http://www.soapatterns.org, 2008

[19] Šlefr Jiří, *Integrační patterny*, 2008

[20] Šlefr Jiří, *Standardy Integrace IS v ČP*, 2008

[21] Vermaas Gero, *Top 10 SOA Pitfalls: #4 - Incorrectly applied Canonical Data Model* , http://blog.xebia.com/2008/06/02/top-10-soa-pitfalls-4-incorrectly-applied-canonical-data-model/, Jún 2009

Dodatok A

Obsah priloženého CD

`models/` – použité UML modely pre Power Designer

`osb/` – osb doména s príkladom z praktickej časti

`thesis/` – text diplomovej práce vo formáte pdf

`workshop/` – workshop projekt s príkladom z praktickej časti